Universidade de Aveiro Departamento de Eletrónica, Telecomunicações e Informática
2018

ARICSON DAVID PINTO DO ROSÁRIO

DATA WAREHOUSE EM TEMPO REAL PARA APOIO AOS TRIBUNAIS DE CABO VERDE

Universidade de Aveiro
2018
Departamento de Eletrónica, Telecomunicações e Informática

ARICSON DAVID PINTO DO ROSÁRIO

DATA WAREHOUSE EM TEMPO REAL PARA APOIO AOS TRIBUNAIS DE CABO VERDE

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia Informática, realizada sob a orientação científica do Doutor Cláudio Jorge Vieira Teixeira, Professor Equiparado a Investigador Auxiliar e do Doutor Joaquim Manuel Henriques de Sousa Pinto, Professor Auxiliar do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro

Dedico este trabalho à minha companheira pelo incansável apoio e aos meus pais por acreditarem e terem investido na minha formação académica.

o júri

presidente

Prof. Doutor Ilídio Fernando de Castro Oliveira
Professor Auxiliar da Universidade de Aveiro

Prof. Doutor André Frederico Guilhoto Monteiro
Professor Auxiliar Convidado do Instituto Superior Miguel Torga

Doutor Cláudio Jorge Vieira Teixeira
Equiparado a Investigador Auxiliar da Universidade de Aveiro

agradecimentos

Quero agradecer aos meus pais, meus irmãos e meus avós por sempre me terem encorajado a acreditar em mim, a ultrapassar obstáculos sempre com respeito pelas outras pessoas, e a lutar pelos meus objetivos. Sempre serei grato por tudo o que fizeram por mim.

Aos Orientadores Cláudio Jorge Vieira Teixeira e Joaquim Manuel Henriques de Sousa Pinto por me terem atribuído este desafio, e pelo conhecimento que foram-me transmitindo ao longo deste percurso

Aos meus colegas do laboratório no IEETA, especialmente ao Gonçalo, pelos Incentivos e por acreditarem que conseguia levar este trabalho a bom porto.

A todos aqueles que de alguma forma contribuíram para a realização deste trabalho.

palavras-chave

Data Warehouse, OLTP, OLAP, ETL, arquitetura, tempo real, sistema de informação, base de dados, ferramentas de visualização.

resumo

As bases de dados transacionais não foram concebidas para responder de forma rápida as consultas que requerem o processamento de grandes volumes de dados. Para solucionar este problema, foi implementada uma arquitetura tendo o *Data Warehouse* em tempo real como componente principal, cujo objetivo é diminuir o tempo de processamento desse tipo de consultas. Esta arquitetura é formada por múltiplos componentes, que realizam várias operações que vão desde a deteção de uma inserção ou alteração de dados nas bases de dados transacionais, até a sua respetiva publicação no *Data Warehouse*, altura em que passam a estar disponíveis para consulta.

keywords

Data Warehouse, OLTP, OLAP, ETL, arquitetura, tempo real, sistema de informação, base de dados, ferramentas de visualização

abstract

Transactional databases weren't designed to respond quickly to queries that require the processing of substantial amounts of data. To solve this problem, an architecture was implemented having the Data Warehouse in real time as main component, whose goal is to reduce the processing time of this type of queries. This architecture consists of multiple components, which perform tasks from the detection of an insertion or change of data in the transactional databases until their respective publication in the Data Warehouse, being available for query from that point on.

Índice

Lista de Ilustrações

Lista de tabelas

Lista de Acrónimos

ACID	*Atomicity, Consistency, Isolation, Durability*
AMQP	*Advanced Message Queuing Protocol*
API	*Application Programming Interface*
BI	*Business Intelligence*
CDC	*Change Data Capture*
DGCI	Direção Geral de Contribuições e Impostos
DJE	Diário de Justiça Eletrónico
DML	*Data Manipulation Language*
DW	*Data Warehouse*
ETL	*Extract, transform, load*
HDFS	*Hadoop Distributed File System*
HTTP	*HyperText Transfer Protocol*
IDE	*Integrated Development Environment*
IUJ	Identificador Único de Justiça
J2EE	*Java 2 Enterprise Edition*
JMS	*Java Message Service*
JVM	*Java Virtual Machine*

LDAP	*Lightweight Directory Access Protocol*
LTS	*Long Term Support*
MQTT	*Message Queuing Telemetry Transport*
MVC	*Model View Controller*
OACV	Ordem dos Advogados de Cabo Verde
OLAP	*Online Analytical Processing*
OLTP	*Online Transaction Processing*
RAM	*Random Acess Memory*
REST	*Representational State Transfer*
RNI	Registo Nacional de Identificação
SIIC	Sistema de Informação e Investigação Criminal
SIJ	Sistema de Informação da Justiça
SIPC	Sistema de Informação do Processo Civil
SIPP	Sistema de Informação Penal
SMS	*Short Message Service*
SOA	*Service Oriented Architecture*
SQL	*Structured Query Language*
STOMP	*Simple Text-Orientated Messaging Protocol*
TIC	Tecnologias da informação e comunicação
TLS	*Transport Layer Security*
TS	*Time-Series*
URL	*Uniform Resource Locator*
VCPU	*Virtual Central Processing Unit*

XML *Extensible Markup Language*

XMPP *Extensible Messaging and Presence Protocol*

YCSB *Yahoo! Cloud Serving Benchmark*

1. Introdução

Ao longo deste capítulo será feita uma apresentação detalhada sobre os motivos, as soluções e o porquê de desenvolver este trabalho. Com este propósito, no enquadramento são expostos os problemas detetados, e depois na motivação são apresentadas de forma generalizada as possíveis soluções. Nesta mesma logica é feita uma descrição detalhada dos objetivos que se pretendem alcançar. De seguida, na metodologia de trabalho são explicados todos os passos seguidos para o desenvolvimento deste trabalho. Posteriormente, no tópico relativo ás contribuições são descritos os ganhos que o desenvolvimento desta solução irá trazer para o sistema de informação da justiça, e para todos os que nele trabalham. Por último, na organização do trabalho é especificado como este relatório se encontra estruturado.

1.1. Enquadramento

Atualmente o sistema judicial em cabo verde encontra-se dividido em 16 comarcas. Neste momento apenas o processo penal encontra-se abrangido pelo Sistema de Informação da Justiça (SIJ), porém nas próximas versões do SIJ espera-se conseguir terminar o que é necessário para suportar também o código civil. O SIJ foi desenvolvido pela Universidade de Aveiro em parceria com diversos intervenientes da justiça de Cabo Verde e serve de instrumento de trabalho, essencialmente para magistrados, procuradores, oficiais de justiça, advogados e órgãos de polícia criminal [1]. Em virtude da quantidade de utilizadores que trabalham no sistema diariamente, e com o número de processos no sistema a aumentar, os pedidos dos utilizadores ao sistema especialmente aqueles que requerem processamento de grandes volumes de dados, tendem a demorar mais tempo para apresentar resultados. Um caso típico onde isto acontece, é na determinação do número de tarefas pendentes para

utilizadores com uma acumulação enorme de processos e tarefas por executar no sistema. Esta demora justifica-se não só pela quantidade de informação a ser processada, mas principalmente pela quantidade de junções de tabelas necessários para responder aos pedidos, visto que neste momento na base de dados principal do SIJ existem cerca de 401 tabelas operacionais. Uma vez que o SIJ foi desenvolvido sobre uma base de dados concebida para o processamento de transações, deste modo, não se consegue dar respostas a esse processamento massivo de dados em tempo real. Por isso, é preciso uma alternativa credível para diminuir esses tempos de execução e que complementarmente possa servir de apoio às tomadas de decisões.

1.2. Motivação

Tendo em consideração as exigências dos desafios apontados, e com a base de dados do SIJ crescendo em termos de dados, onde a rapidez com que se acede a informação pode ser um fator diferenciador de qualidade, faz com que aumente a demanda por soluções de consulta de dados mais eficientes. Para responder a estas demandas é necessário que as informações requisitadas estejam disponíveis para consulta quase que de forma imediata, a partir do momento em que são inseridos nas bases de dados do SIJ, independentemente da quantidade de dados a ser processados. Contudo isto nem sempre acontece, porque inicialmente o SIJ não foi projetado para processar essas quantidades de informações num espaço de tempo tão curto, e consequentemente algumas tarefas específicas do sistema consomem mais tempo do que era espectável para retornarem os resultados das tarefas em questão. Deste modo, devido essencialmente à necessidade de os utilizadores terem os seus pedidos ao sistema respondidos no menor tempo possível, um *Data Warehouse* em tempo real pelas suas características, apresenta-se como a escolha mais acertada para este tipo de desafio. Assim sendo, pretende-se implementar uma arquitetura de processamento de dados em tempo real, onde a partir do momento em que for detetado qualquer tipo inserção ou alteração de dados na base de dados operacional do SIJ, essa alteração seja refletida no *Data Warehouse* num curto espaço de tempo (segundo ou minutos), passando a estar disponível para eventuais consultas. Este tipo de infraestrutura ao ser incorporado no SIJ, passará a ser utilizado diariamente em segundo plano pelos utilizadores, respondendo às consultas efetuadas a base de dados no menor tempo possível.

Além disso, aproveitando o facto de esta infraestrutura estar implementada, e utilizando uma ferramenta de visualização podem-se efetuar análises dos dados dos tribunais de uma forma mais apropriada e eficiente. Isto facilita o trabalho de interpretação dos dados por parte dos decisores, mais propriamente dos membros do conselho superior e de todos os órgãos responsáveis pelas tomadas de decisões, e consequentemente com isso, espera-se aumentar as taxas de sucesso na luta e prevenção à criminalidade.

1.3. Objetivos

Dado os constrangimentos apresentados, pretende-se criar num ambiente separado da produção, uma arquitetura desenhada e adaptada para o SIJ, que seja escalável, altamente disponível, com um consumo de fluxo de dados em tempo real e idealmente sem custos. Esta arquitetura tem como base ou fundação o *Data Warehouse*, utilizado principalmente com o intuito de diminuir o tempo de execução das tarefas no SIJ que requerem processamento de grandes quantidades de informações, utilizando várias técnicas de processamento de dados, como indexação, agregações e pré-cálculos. Desse modo, propõe-se diminuir para uma escala de segundos, o tempo necessário para que o *Data Warehouse* esteja atualizado com dados recentes, ou seja reduzir ao mínimo possível o espaço temporal, entre inserir ou alterar uma informação na base de dados de processamento de transações *on-line* (OLTP), e colocá-la disponível para consulta no *Data Warehouse*. Para este propósito, também será construído um conjunto de funções e interfaces programáticas (API) para acesso aos dados armazenados no *Data Warehouse*, a partir do sistema de informação de justiça. Isto permitirá ao SIJ utilizar a base de dados de processamento analítico *on-line* (OLAP) do *Data Warehouse*, para responder aos pedidos de consulta de dados mais exigentes em termos de processamento, em detrimento de serem executadas diretamente na base de dados OLTP do SIJ, o que exigiria um tempo maior de processamento.

Além disso, a aplicação desta nova arquitetura permitirá rentabilizar e otimizar a distribuição de carga na base dados do SIJ, na medida em que tarefas com maior carga de dados e que demoram mais tempo a executar, serão transferidas para a base de dados OLAP, libertando assim a base de dados OLTP para outras tarefas.

Outra finalidade desta solução está relacionada com a possibilidade de permitir análises com dados em tempo real, através de uma ferramenta de visualização OLAP. Esta ferramenta de visualização deverá entre outras análises, permitir comparar dados de crimes por datas, ilhas, cidades ou regiões, comarcas, tribunais, entre outras variáveis, bem como criar relatórios com esses mesmos dados. Também deverá possibilitar identificar onde ocorrem certos tipos de crimes com mais frequência. Isso poderá ser relevante, visto que pode mostrar que áreas são mais propícias a ocorrências de determinados crimes, permitindo assim tomar medidas no sentido de melhorar a eficácia no combate a estes crimes nessas áreas.

1.4. Metodologia de Trabalho

Ao longo deste projeto foi adotado a metodologia de investigação denominado "*The Engineering Design Process*" [1]. A metodologia consiste numa série de passos seguidos pelos engenheiros para encontrar uma solução para um problema. As etapas a seguir dessa metodologia são variadas, nomeadamente a definição do problema, pesquisa aprofundada sobre o tema, especificação de requisitos, *brainstorming* de soluções, escolha da melhor solução, trabalho de desenvolvimento, construção do protótipo, teste, e por último, se a solução não for encontrada, são feitas as devidas alterações e o processo é realizado novamente a partir de onde o problema foi identificado [1].

De acordo com a metodologia seguida, primeiramente definiu-se o problema: "Como agilizar o processamento de grandes quantidades dados para responder as consultas dos utilizadores do SIJ em tempo real?". Depois, seguiu-se para a pesquisa e recolha aprofundada do estado da arte sobre *Data Warehouse* em tempo real e ferramentas de visualização OLAP. Com isso, elaborou-se uma versão protótipo de um documento onde foi feita a especificação dos requisitos e descrito tudo o que havia sido recolhido sobre o tema de dissertação. No próximo passo, foi feito um levantamento das soluções que se adequam aos requisitos anteriormente especificados, e dessas soluções foi escolhida a que dava melhores garantias de desempenho e idealmente sem custos. Em seguida foi feito o desenvolvimento dos vários blocos que compõem a arquitetura proposta neste trabalho. Cada um destes blocos foi testado exaustivamente com dados fictícios e posteriormente reagrupados para formar a arquitetura do sistema. Com a arquitetura completa foram feitos

testes tanto com dados fictícios como também com dados reais. Por último, caso a solução não funcionasse tal como esperado ou fosse detetado algum erro, então o processo era repetido novamente a partir do ponto onde o problema pudesse ser corrigido.

Após a avaliação da solução criada, foram analisados os resultados para aferir se estavam de acordo com o pretendido e se era vantajoso ou não uma nova iteração para melhorar componentes específicas da arquitetura.

1.5. Contribuição

Nesta dissertação é detalhadamente descrito como desenhar e implementar uma arquitetura de um sistema de *DataWarehouse* em tempo real, que permita essencialmente agilizar o processo de consulta dos dados, desde a captura dos dados no momento em que foram introduzidos numa base de dados operacional, até o instante em que são colocados disponíveis para serem consultados pelo utilizador.

1.6. Organização da Dissertação

Este trabalho está dividido em sete capítulos, sendo o presente capítulo destinado ao enquadramento do problema e à contribuição da solução conseguida nesta dissertação. No segundo capítulo são descritos os aspetos gerais em termos de objetivos e arquitetura do SIJ. Também é explicado todo o processo de gestão de tarefas pendentes dos utilizadores ao longo da tramitação de um processo. O terceiro capítulo é destinado ao estado de arte, descrevendo conceitos teóricos sobre os componentes da arquitetura proposta. No quarto capítulo são apresentados o desenho e a arquitetura do sistema implementado, através dos quais foram descritas o funcionamento do processo ETL (extração, transformação e carregamento de dados), os componentes do serviço de consumo de dados em tempo real, e a estrutura da interface desenvolvida entre o *Druid* e o SIJ. Neste capítulo estão disponíveis vários diagramas para ajudar a entender todo o processo. No quinto capítulo são expostos todos os detalhes de implementação, onde é explicado como cada componente da arquitetura foi implementado e configurado de forma a obter o melhor desempenho possível no processamento dos dados provenientes do SIJ. No sexto capítulo são apresentados os resultados dos testes efetuados a cada um dos componentes da arquitetura separadamente. Sendo posteriormente calculado para uma das fontes de dados,

um valor representativo da latência para toda a infraestrutura, agregando os valores obtidos para cada componente da arquitetura. Por último, no capítulo sete são expostas as conclusões, problemas encontrados e trabalhos futuros.

2. Sistema de Informação da Justiça de Cabo Verde

O presente capítulo apresenta os conceitos gerais relativos ao funcionamento interno do SIJ, e a descrição de como ele se integra com outros sistemas. Ainda neste capítulo é apresentado de forma detalhada, a estratégia utilizada na determinação de tarefas pendentes para cada utilizador ao longo da tramitação de um processo.

2.1. Visão Geral do SIJ

O Sistema de Informação da Justiça de Cabo Verde (SIJ) é um projeto baseado na *web*, desenvolvido essencialmente para a tramitação eletrónica dos processos nos tribunais de Cabo Verde. Este sistema permite a desmaterialização total dos processos, em que toda a tramitação relativa a um processo é efetuada no sistema, ou a desmaterialização parcial dos processos, em que apenas parte da tramitação do processo é efetuada no SIJ. Com este sistema, diferentes utilizadores podem interagir em simultâneo com os processos tramitados, dependendo da fase judicial em que o processo se encontrar. Fundamentalmente, os utilizadores do SIJ são, os oficiais de justiça, os juízes, os procuradores, os advogados e os membros dos conselhos superiores [2]–[4].

Cabo Verde é um país com 9 ilhas habitadas, em que muitas vezes se impõe a deslocação dos magistrados entre comarcas, geralmente em ilhas distintas, pelo que este sistema foi concebido para que pudesse ser acedido em qualquer local com internet, desde que, garantidas as condições de segurança exigidas nos tribunais [5]. Assim sendo, este sistema possui mecanismos de segurança tanto através de autenticação, como também utilizando certificados que garantem acesso ao sistema apenas para os utilizadores devidamente autorizados (Figura 1).

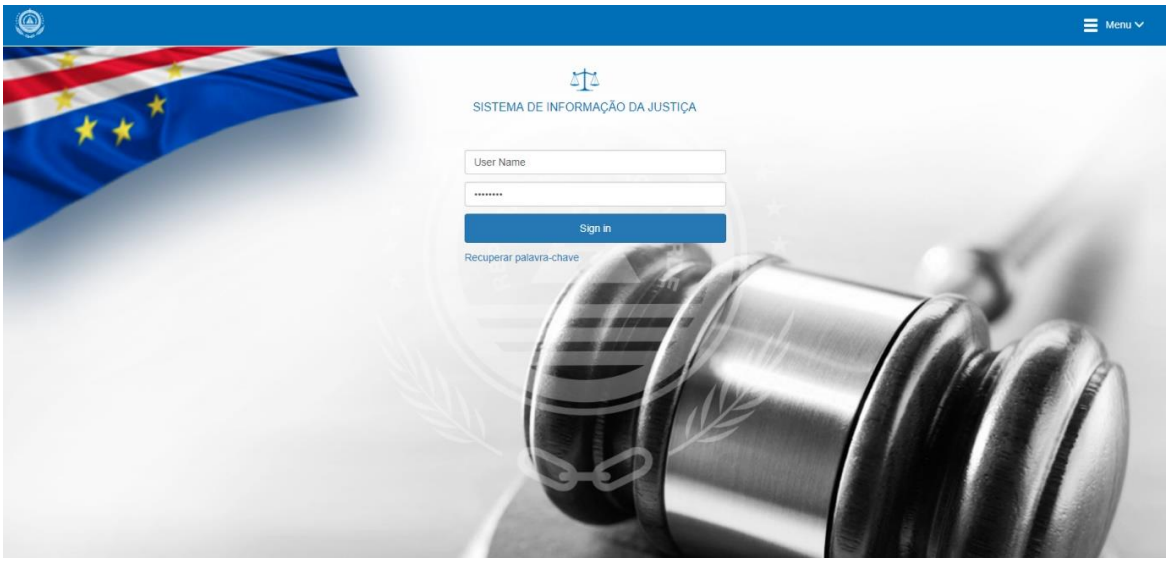

Figura 1 - Página Inicial do SIJ

O SIJ foi modelado de acordo com a natureza jurídico-penal dos crimes e tramitação processual previstas no Código Penal e de Processo Penal de Cabo Verde. Para definição das funcionalidades do sistema, foi constituída uma comissão de acompanhamento composta por juízes, procuradores do Ministério Publico, oficiais de justiça e advogados. Como intermediários entre essa comissão de acompanhamento e a equipa de desenvolvimento, foi criada uma equipa de modelação, composta por docentes da área de informática e por juristas [5]. Isto, para que o sistema fosse potenciado o mais próximo possível daquilo que os utilizadores pretendiam.

2.2. Integração com outros sistemas

Tal como apresentado em [6], o SIJ foi projetado para ser composto por vários componentes, entre os quais o Sistema de Informação Penal (SIPP), Sistema de Informação do Processo Civil (SIPC), Diário de Justiça Eletrónico (DJE), Ordem dos Advogados de Cabo Verde (OACV), Portal Operacional de Tecnologias da Informação, Identificador Único de Justiça (IUJ). Além disso, o SIJ permite a comunicação com outros sistemas, como a Direção Geral de Contribuições e Impostos (DGCI), o Registo Nacional de Identificação (RNI), Sistema de Informação e Investigação Criminal (SIIC), entre outros. A Figura 2 é uma ilustração com todos os componentes do SIJ, distinguindo que componentes já foram desenvolvidas e os que faltam desenvolver.

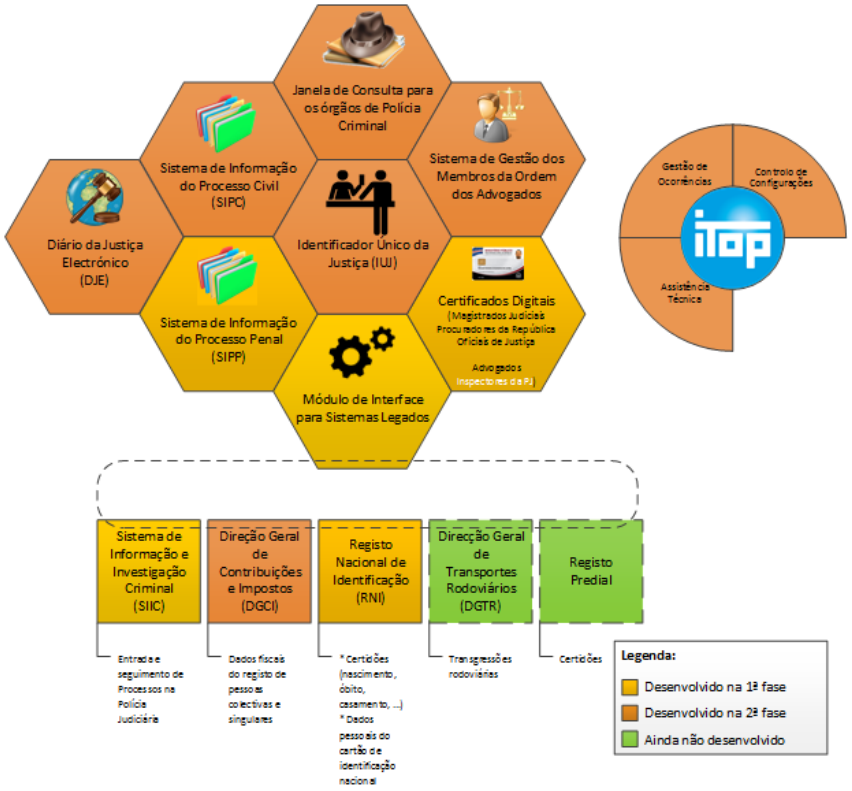

Figura 2 - Componentes do SIJ (equipa de desenvolvimento)

Neste momento, o SIPP é aquele que se encontra numa fase mais adiantada de desenvolvimento, tendo já oficialmente começado a sua integração nos tribunais de Cabo Verde. Portanto, numa fase inicial, este trabalho de implementação de *Data Warehouse* em tempo real, esta sendo concebido para ser integrado apenas ao SIPP.

2.3. Arquitetura do Sistema de Informação de Processos Penais

A arquitetura do Sistema de Informação de Processos Penais (SIPP), é dividida em 4 camadas, uma aplicação *web* (*interface* com o utilizador), os serviços, a lógica de negócios, e a base de dados, de acordo com as tarefas e funções que cada um desempenha (Figura 3).

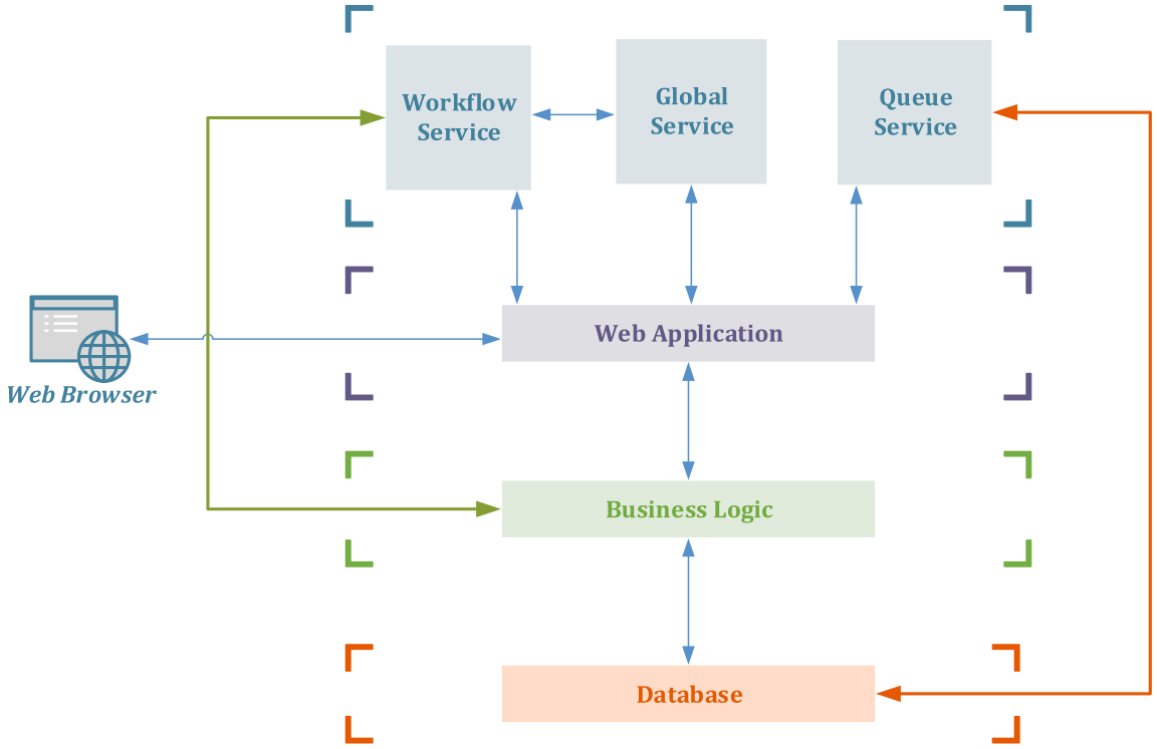

Figura 3 - Arquitetura do SIPP [6]

A aplicação *web* é a *interface* dos utilizadores com o sistema, onde podem realizar um conjunto alargado de ações sobre os processos da justiça de cabo verde, entre os quais a interação com processos judiciais, gestão de notificações, mensagens trocadas entre utilizadores, acesso a documentos legais, gestão de anotações, calendarização de tarefas, entre muitos outros [7].

Para que a camada de *interface* consiga aceder aos dados armazenados na base de dados, e efetue todas as operações e transformações necessárias sobre esses mesmos dados, é utilizada a camada de lógica de negócios. Esta camada é o núcleo ou a parte central do sistema, o qual permite que a aplicação *web* e o serviço de *workflow* partilhem o mesmo código base, e onde é feito todo o processamento dos dados provenientes da base de dados, a lógica de validação, a lógica de segurança, entre outros [7].

A camada de base de dados está implementada sobre o sistema de gestão de base de dados *SQL SERVER 2008 R2* [8]. A ligação entre a camada de base de dados e a camada de lógica de negócios é feita através do *Entity Framework* [9].

Por último, existe a camada de serviços, composta por 3 tipos de serviços, *workflow*, global e de base de dados (*Queue*). Estes serviços realizam a comunicação com a camada da aplicação *web* através de uma arquitetura orientada a serviços (SOA). O serviço de *workflow* faz a gestão de todos os procedimentos envolvidos nos processos judiciais. Ele tem como principal função a gestão de todos os procedimentos envolvidos nos processos judiciais, como a gestão de todos os estados, transações e interações que podem ocorrer em cada processo judicial. Também entre muitas outras funções, é da responsabilidade deste serviço, a definição dos perfis de utilizadores que podem interagir com um processo numa determinada fase, e os tipos de decisões possíveis em cada fase do processo [7].

O serviço de base de dados permite gerir a execução de tarefas na base de dados que antes provocavam problemas de *deadlocks*. Para evitar bloqueios de acesso as tabelas, foi desenvolvida uma estrutura em forma de serviços *Windows*, o qual através de filas de espera possibilita a execução das tarefas de forma sequencial. Ao passo que, o serviço global permite gerir funcionalidades globais do sistema, como o serviço de mensagens curtas (SMS), persistência ou sincronização [6].

2.4. Determinação das Tarefas pendentes no SIJ

Além de várias outras funcionalidades, a aplicação web do SIJ possui uma secção onde são apresentadas as tarefas pendentes do utilizador, dependendo do cargo pelo qual desempenha. Basicamente, uma tarefa pode traduzir-se quer na existência de mensagens eletrónicas novas, quer na existência de trabalhos pendentes em processos associados ao utilizador (Figura 4). O utilizador tem a possibilidade de visualizar e interagir com as suas tarefas dando prosseguimento a tramitação eletrónica dos processos sobre os quais esta trabalhando.

Tarefas

Tipo	Descrição	Data	Data Limite
	Distribuição de processo por Rogério Alcides Fernandes [Chefe Sec. MP]	08-01-2018	15-01-2018
	Entrada de novo Auto por Rogério Alcides Fernandes [Chefe Sec. MP]	08-01-2018	09-01-2018
	Entrada de novo Auto por Rogério Alcides Fernandes [Chefe Sec. MP]	04-01-2018	05-01-2018
	Entrada de novo Auto por Rogério Alcides Fernandes [Chefe Sec. MP]	10-10-2017	11-10-2017
	Processo n.º 2/2017 a aguardar explicação	10-10-2017	11-10-2017
	Entrada de novo Auto por Rogério Alcides Fernandes [Chefe Sec. MP]	10-10-2017	11-10-2017
	Processo n.º 4/2017 a aguardar explicação	10-10-2017	11-10-2017
	Distribuição de processo por Rogério Alcides Fernandes [Chefe Sec. MP]	14-07-2017	21-07-2017
	Entrada do despacho n.º 14-07-2017	15:41 - Processo n.º 3/2017 por lazaro lopes rocha [Procurador]	14-07-2017	15-07-2017
	Entrada de novo Auto por Rogério Alcides Fernandes [Chefe Sec. MP]	14-07-2017	15-07-2017
	Distribuição de processo por Rogério Alcides Fernandes [Chefe Sec. MP]	20-06-2017	27-06-2017
	Entrada de novo Auto por Rogério Alcides Fernandes [Chefe Sec. MP]	20-06-2017	21-06-2017

Figura 4 - Exemplo de como as tarefas pendentes são apresentadas no SIJ

Em específico, trabalhos pendentes como processos a aguardar explicação, despachos a aguardar tramitação na secretaria, e requerimentos a aguardar despachos, são as tarefas que habitualmente ocorrem com maior frequência, sendo os procedimentos que os despoletam, os mais comumente utilizados pelos atores judiciais em fases distintas, na generalidade dos processos.

Correntemente, existe mais do que uma forma de se dar início a um processo no SIJ, como por exemplo por meio da emissão de um despacho de separação de processos, mas normalmente o modo mais utilizado é através de um auto de denúncia ou detenção em flagrante delito. Ao longo do ciclo de vida de um processo, este pode englobar muitas mudanças de fase e participação de vários intervenientes da justiça que o fazem desenrolar até à determinação de uma decisão ou arquivamento.

Ao introduzir um auto no sistema, este é atribuído automaticamente a um ou mais oficiais de justiça dependendo do tribunal selecionado, e enquanto não for feita a explicação dos factos do respetivo auto, este aparecerá representado na área de tarefas desses oficiais como um processo a aguardar explicação. Este auto deixará de estar pendente na lista de tarefas apenas quando for feito o cumprimento do auto por um dos oficiais. Assim sendo, o sistema designa este oficial como o autor da explicação do auto. Logo, todo o auto com um autor de explicação deixa de estar pendente. Em decorrência, e em condições normais, após o chefe da secretaria indicar qual a equipa do Ministério Público a que caberá instruir o processo, a distribuição a um dos Magistrados dessa equipa é feita de forma automática. Cada equipa é composta por pelo menos um procurador, o qual dá continuidade a tramitação do processo, normalmente através da emissão de despachos. Existem vários

tipos de despacho, contudo todos dão origem a tarefa pendente designada de despacho a aguardar tramitação na secretaria, na lista de tarefas do procurador. Esta tarefa também estará disponível no portfólio de tarefas dos oficiais cujo acesso foram concedidos pelo procurador. Após ser efetuado o cumprimento do despacho mediante a explicação dos factos, por parte de um dos oficiais com acesso, ou pelo próprio procurador, esta tarefa deixará de estar pendente, sendo removido da lista de tarefas.

Além disso, o procurador tem a possibilidade de remeter o processo para o tribunal, mediante a emissão de um requerimento de promoção de julgamento em processo comum e perante tribunal singular. Ao dar entrada nos tribunais o processo muda da fase de instrução para marcação de julgamento. Desse modo é adicionado a lista de tarefas do Juiz a tarefa pendente de requerimento a aguardar despacho. Caso o Juiz não encontre nenhuma discrepância que coloque em causa o seguimento do processo dentro dos trâmites legais, é da responsabilidade deste determinar a marcação do julgamento mediante um despacho emitido no sistema. Similarmente ao despacho anterior, o cumprimento deste despacho pode ser feito por um oficial do juízo com o devido acesso concedido pelo juiz ou pelo próprio juiz. Após a explicação deste despacho o processo passa para fase de aguardar marcação de julgamento, até a data em que foi marcada o julgamento. Terminado este período o processo entra na fase de julgamento. Dependendo da complexidade de um processo, a tramitação pode envolver outros procedimentos e várias outras fases, como por exemplo trânsito em julgado que é depois da fase de julgamento. Portanto, conforme a especificidade de um processo, a tramitação pode ocorrer a vários níveis, passando por diversas fases, em que vários atores judiciais vão interagindo com o processo.

O processamento e construção das listas de tarefas pendentes, por utilizador, é um procedimento complexo e que exige, neste momento, bastantes recursos. As tarefas pendentes não se restringem às informações apresentados anteriormente. Ainda assim, as tarefas apresentadas representam uma parte considerável do conjunto total de tarefas e, acima de tudo, permitem ter uma ideia da complexidade em questão.

3. Estado de Arte

Com o crescimento da internet e a utilização massiva de tecnologias de informação, a quantidade de dados gerados e disponibilizados tem crescido de forma exponencial. Um estudo intitulado "*A Universe of Opportunities and Challenges*", desenvolvido pela consultoria EMC [10], aponta que até 2020 existe a perspetiva que o volume de dados alcance a ordem dos 40.000 Exabytes, ou 40 triliões de Gigabytes. Este crescimento expressivo dos dados justifica-se sobretudo pelo aumento da necessidade de informação, e pela evolução das tecnologias da informação e comunicação (TICs).

Desse modo, gradualmente mais organizações em todo o mundo têm-se consciencializado da necessidade de utilizarem o *Business Intelligence* (BI) para suporte às tomadas de decisões. O BI inclui a infraestrutura, as aplicações, as ferramentas e as melhores práticas, que permitem a análise de uma vasta quantidade de informação (*Big Data*) para melhorar e otimizar as decisões e o desempenho [11]. Os utilizadores são cada vez mais exigentes para que dados atualizados estejam disponíveis onde e quando precisam. No entanto não é viável executar relatórios de grandes quantidades de dados sobre os sistemas projetados para processamento de transações, arriscando-se a sobrecarregar a base de dados. Uma solução comum é utilizar outra base dados para o qual os dados de produção são transferidos, e só então nesta base de dados executar relatórios. Mas para que essa solução funcione de acordo com o pretendido, é preciso que as atualizações no servidor de análise de dados sejam feitas o mais rapidamente possível.

Para aplicar estes conceitos no SIJ importa, portanto, e em primeiro lugar, estudar o que já existe sobre o tema deste trabalho, desde a captura de dados atualizados, ferramentas de

integração de dados, *Data Warehouse* em tempo real, até ferramentas de visualização de dados.

3.1. Captura de alterações de dados

O CDC (*Change Data Capture*) é uma abordagem de integração de dados que identifica e captura alterações (*insert*, *update* e *delete*) feitas a base de dados. Estas alterações podem ser aplicadas a outros repositórios de dados ou disponibilizadas em formato consumível através de uma ferramenta de extração, transformação e carregamento de dados (ETL), ou utilizando outros tipos de ferramentas de integração de dados [12]. Este mecanismo de captura de dados alterados elimina a necessidade de atualização massiva dos dados, permitindo o carregamento incremental ou a transmissão em tempo real das últimas alterações para o *Data Warehouse*. Ao carregar apenas os dados mais recentes, o CDC limita o impacto nas fontes de dados operacionais, reduz o tempo de processamento necessário e os custos de recursos do *Data Warehouse*, ao mesmo tempo que permite a integração contínua de dados.

Tradicionalmente, para captura de dados alterados ou recentemente inseridos na base de dados, eram utilizados técnicas como diferenciação de tabelas, seleção de valor de mudança e *triggers*. No entanto, essas técnicas podem ser ineficientes ou intrusivas e tendem a colocar uma sobrecarga substancial nos servidores OLTP [12]. Como forma de resolver este problema, o mecanismo de captura de dados alterados (CDC) foi introduzido em bases de dados como o *SQL Server*, *MySQL*, *PostgreSQL* e Oracle. Tendo em consideração que a fonte de dados do SIJ é inteiramente implementada num servidor de base de dados SQL, portanto, irão ser consideradas apenas as soluções de CDC com suporte a este tipo base de dados

3.1.1. CDC do SQL SERVER

O servidor de base de dados SQL possui um mecanismo integrado para captura de dados alterados através de *log*. Este tipo de CDC baseado em *log* utiliza um processo em segundo plano para verificar registos de transações, fazendo com que o impacto no desempenho dos servidores de origem seja mínimo. Qualquer transação feita na base de dados é registada no *log* de transações da base de dados do servidor do SQL. Ao ativar o CDC sobre uma

base de dados, automaticamente é criado um agente SQL para a tarefa de captura de dados nessa mesma base de dados. A tarefa de captura lê o log de transações do *SQL Server* para recuperar todas as modificações feitas as tabelas monitoradas. A medida que essas alterações são lidas, vão sendo gravadas em tabelas criadas pelo CDC para manter os dados capturados por um tempo previamente configurado. Depois, dependendo do intervalo de tempo configurado, o processo de captura lê o *log* e adiciona os dados das alterações nas tabelas associadas criadas pelo CDC [13]. Opcionalmente, as informações capturadas e os metadados associados podem ser transferidas para um aplicativo ETL ou outra ferramenta de integração dos dados, que por sua vez carrega incrementalmente os dados atualizados para um *Data Warehouse* ou *Data Mart*.

3.1.2. Attunity Replicate

O *Attunity Replicate* é uma ferramenta proprietária para replicação e consumo de dados baseado em *log*. Esta ferramenta utiliza uma *interface* bastante intuitiva, e automatiza o processo de replicação da origem até ao destino, eliminando a necessidade de codificação manual do processo ETL, ou conhecimento abrangente de base de dados [14]. Com uma tecnologia CDC de última geração, além de suportar a movimentação de dados em uma ampla variedade de base de dados heterogéneos, incluindo o *SQL Server* e plataformas de *Big Data*, também consome e carrega dados de uma maneira rápida e eficiente, sem gerar sobrecarga desnecessária nos sistemas de origem ou de destino. Isto graças à arquitetura *zero-footprint* do *Attunity Replicate*, o qual foi projetado para eliminar a sobrecarga desnecessária nos sistemas de missão crítica, visto que o leitor de *log* pode ser instalado no servidor de replicação, de forma a obter o mínimo impacto no servidor da fonte de dados [15]. Além disso, a arquitetura modular do *Attunity Replicate*, oferece suporte a ambientes de grande volume de dados e em constante mudança, permitindo dimensionar o *Attunity Replicate* conforme os requisitos de dados organizacionais forem aumentando.

3.1.3. Resumo

O CDC do servidor SQL é o mais adequado dado os requisitos deste trabalho. Trata-se de uma ferramenta que não representa nenhum custo para a sua implementação, uma vez que a base de dados do SIJ já se encontra montada sobre o *SQL SERVER*, sendo que o CDC já

se encontra integrado no SQL *SERVER* desde a versão de 2008. A Tabela 1 apresenta o resumo das características anteriormente especificadas.

	CDC SQL SERVER	*Attunity Replicate*
Baseado em *Log*	X	X
Integrado no *SQL SERVER*	X	
Suporta vários tipos de Base de dados		X
Tem um custo de aquisição associado		X
Solução Adotada	X	

Tabela 1 - Comparação entre o CDC do servidor SQL e do Attunity Replicate

3.2. Extração, Transformação e Carregamento de dados

Um fator considerado chave para o sucesso de qualquer projeto de *Data Warehouse* é o processo de extração, transformação e carregamento dos dados (ETL), pois a qualidade dos dados inseridos influência drasticamente os resultados apresentados na análise de dados [16]. Normalmente o ETL é o que gasta mais tempo no processo de atualização do *Data Warehouse*. Algumas publicações de trabalhos tal como [17], se enquadram nos sistemas OLAP "quase em tempo real", pelo que começaram por tentar resolver este problema, minimizando ao máximo o tempo despendido no ETL. Outro trabalho realizado neste sentido é o [18], onde empregam um modelo de matriz multidimensional extensível para efetuar atualizações ao cubo de dados de forma incremental. Porém, devido aos problemas de desempenho significativos com *Data Warehouse* em larga escala, estas abordagens não são consideradas como completamente em tempo real.

Este cenário vem mudando, com maior preponderância nestes últimos anos, onde tem surgido cada vez mais ferramentas no sentido de facilitar a integração de dados entre sistemas diferentes com o mínimo de desfasamento possível na atualização dos dados.

3.2.1. Apache Nifi

O *Apache NiFi* é uma ferramenta de integração de dados concebida para automatizar o fluxo de dados entre sistemas. A partir do momento em que os dados são consumidos,

automaticamente são representados como *FlowFile* dentro do fluxo de dados do *Apache Nifi*, sendo que um *FlowFile* é basicamente um conjunto de registos de dados originais com metadados associados.

Efetivamente, o que torna esta ferramenta tão atrativa é o facto de fornecer uma plataforma que permite entre outras funções, coletar, selecionar, analisar e atuar sobre os dados em tempo real, localmente ou na nuvem [19]. O código é abstraído atrás de uma *interface* de arrastar e soltar, permitindo a implementação de todo o fluxo de dados de uma maneira mais fácil e eficiente. Em contrapartida, quando não for possível construir determinado fluxo sem recurso a programação, neste caso o *Apache Nifi* oferece a possibilidade de utilizar processadores que suportam codificação manual, utilizando linguagem como *Clojure*, *ECMAScript*, *Groovy*, *Lua*, *Phython* e *Ruby*.

Na prática utiliza-se o *Apache NiFi* sobretudo quando as informações precisam ser processadas por meio de uma série de etapas incrementais, sendo que é possível ao longo de cada uma destas etapas fazer a depuração de erros, conseguindo corrigi-los sem a necessidade de parar todo o fluxo. Esta ferramenta disponibiliza funcionalidades para facilitar e agilizar o processo de construção do fluxo de dados, como a possibilidade de agrupar processos, de configurar e reutilizar serviços específicos e de verificar a proveniência de dados [19]. Com efeito, o facto de permitir configurar controladores de serviços, como o JDBC para conexões a base de dados, com o intuito de serem reutilizados em diferentes processadores, facilita e torna mais eficiente a construção do fluxo de dados. Também a proveniência dos dados é extremamente útil na medida em que a qualquer instante consegue-se obter o histórico de onde e como cada *Flowfile* foi executado, permitindo ao mesmo tempo visualizar o conteúdo e metadados associados como o tempo de execução [20].

3.2.2. StreamSets Data Collector

O *StreamSets Data Collector* é outro caso típico de um mecanismo para rotear e processar dados. Sendo definido como uma ferramenta de consumo de dados de baixa latência, que permite criar *pipelines* de entrada de dados contínuos utilizando uma *interface* de arrastar e soltar em um ambiente de desenvolvimento integrado (IDE) [21]. Para além da *interface* de arrastar e soltar, esta ferramenta também permite utilizar linguagens de programação como

o *Python* e *JavaScript*, em processadores especificamente concebidos para serem utilizados com estas linguagens.

Para definir o fluxo de dados para o *Data Collector* é preciso configurar um *pipeline*. À medida que os dados passam pelo *pipeline* é possível inspecioná-los e visualizar estatísticas em tempo real sobre os dados. O *Data Collector* é utilizado principalmente como um canal para fluxos de dados, no qual os fluxos podem ser coletados, movidos e processados no caminho para os seus destinos, sem a necessidade de nenhuma conversão do formato de dados. Os processadores no *Streamsets* trocam registos, isto porque todos os dados consumidos na origem, são automaticamente convertidos no formato padrão, orientado a registos. Também outro dos pontos que vale a pena destacar sobre o *StreamSets*, é o facto de processar dados binários, o que permite aos processadores encaminhar dados de forma mais eficiente com sobrecarga mínima [21].

3.2.3. Resumo

As ferramentas de integração de dados apresentadas, não se diferenciam muito entre si no que diz respeito as funcionalidades principais, mas é necessário escolher uma delas. Sendo que, ambas são ferramentas *open-source*, e possuem uma *interface* de utilizador com inúmeras funcionalidades para tornar o processo de construção de fluxos de dados mais intuitivo, rápido e flexível. Todavia existem algumas diferenças entre elas, abaixo apresentadas na Tabela 2.

	Apache Nifi	*Data Collector*
Interface de arrastar e soltar	X	X
Suporta mais do que uma linguagem de programação	X	X
Depuração de erros	X	X
Configuração e reutilização de serviços específicos	X	
Agrupar Processos ou fluxo de dados por função	X	
Solução Adotada	X	

Tabela 2 - Comparação entre o Apache Nifi e DataCollector.

3.3. Brokers de Mensagens

As Aplicações de *software* modernas raramente existem de forma isolada e para que dois aplicações se comuniquem entre si, eles devem primeiro definir uma *interface*. Definir essa *interface* envolve selecionar o protocolo, como por exemplo REST [22], XMPP [23], MQTT [24], STOMP [25], ou AMQP [26], e que os sistemas envolvidos concordem quanto ao formato das mensagens trocadas [27].

Atualmente é comum consumir informações fornecidas por outras aplicações, o que acentua ainda mais a necessidade de se utilizar ferramentas de integração para interligar sistemas. Um *broker* de mensagens é o exemplo de um tipo de ferramenta de integração, o qual traz muitos benefícios, como desacoplamento entre produtores e consumidores, roteamento de mensagens para um ou mais destinatários, conversões de protocolo e formato de mensagens, melhor controlo e gestão das mensagens, escalabilidade, garantia de entrega, confiabilidade e segurança [27]. Todavia, isto acarreta um custo de manutenção e uma maior latência na transmissão dos dados, mas que normalmente não coloca em causa a utilização desses intermediários de mensagens.

3.3.1. Apache Kafka

O Apache *Kafka* [28] é uma plataforma distribuída de mensagens e *streaming*, originalmente projetado pelo *LinkedIn* para contornar algumas das limitações dos *brokers* de mensagens tradicionais [29], tornando-se mais tarde um projeto da Apache. Inicialmente, o *Kafka* tinha suporte apenas para o Java, mas neste momento suporta várias linguagens de desenvolvimento. Nesta plataforma toda a interação entre produtores e consumidores de mensagens é feita por intermédio de tópicos. Assim, um produtor pode publicar mensagens para um ou mais tópicos. Um tópico é basicamente uma categoria para a qual as mensagens são publicadas.

O *Kafka* depende do sistema de ficheiros (em disco) para armazenar e colocar mensagens em cache. O período pelo qual as mensagens são mantidas no armazenamento de dados persistente é limitado, contudo este parâmetro é configurável, podendo ser ajustado na medida das necessidades. As mensagens publicadas podem ser divididas em partições diferentes e depois armazenadas num conjunto de servidores designados de *brokers*. Uma

partição é uma sequência ordenada e imutável de registos, onde cada registo é identificado através de um número de ID sequencial, designado de *offset* que identifica exclusivamente cada mensagem ou registo dentro da partição [30]. Isto permite que as mensagens sejam consumidas na mesma ordem em que foram publicadas, além de manter um registo de quais foram as mensagens consumidas e por qual consumidor. Tal é necessário visto que os consumidores podem subscrever um ou mais tópicos dos *brokers*, e necessitam de consumir as mensagens subscritas de forma ordenada. Além disso, o *kafka* ao contrário da maioria dos outros *brokers*, mantém as mensagens armazenadas mesmo após serem consumidas, por um período configurável (com um valor por padrão de 168 horas). Isto torna esta ferramenta propícia para desenvolvimento de aplicações, sobretudo de *streaming* de dados, aproveitando os recursos nativos do *kafka* para oferecer paralelismo de dados, coordenação distribuída, tolerância a falhas e simplicidade operacional [31].

3.3.2. Apache ActiveMQ

O *ActiveMQ* [32] é uma implementação *open-source* do JMS 1.1, como parte da especificação do J2EE 1.4. Ele suporta vários protocolos e linguagens de programação do lado cliente como C, C++, C#, Ruby, Python, Perl, e PHP, ao mesmo tempo que facilita a utilização dos padrões de integração e de muitos outros recursos avançados [33]. Este *broker* de mensagens, a semelhança do *Kafka* é baseada no conceito de *publish/subcribe*, sendo equivalente a arquitetura cliente-servidor. Dentro da mesma base de funcionamento desta arquitetura, pode-se incorporar *brokers* do lado cliente, ou seja, localmente, fazendo com que a comunicação entre o cliente e o servidor (*broker*) seja maioritariamente feito dentro da mesma máquina virtual do java (JVM), sem utilizar a rede. Isto traz benefícios em termos de desempenho, reduzindo a latência de comunicação entre cliente e servidor. Não importa o que aconteça as conexões físicas ou ao serviço de destino, a mensagem será entregue ao consumidor, desde que não haja perda catastrófica para o armazenamento de dados persistente do *broker* durante a transmissão da mensagem. Além disso, permite que *clusters* baseados em pares (*Peer-to-Peer*) sejam criados onde não há servidores, apenas clientes se conectando [34].

3.3.3. RabbitMQ

O *RabbitMQ* [35] é um *broker* de mensagens *open-source*, que permite às aplicações conectarem-se utilizando uma variedade de diferentes protocolos abertos e padronizados, como AMQP, STOMP, MQTT, e WebSockets/Web-Stomp [36]. Ele foi originalmente desenvolvido para suportar o AMQP, cujo protocolo é definido como um padrão aberto para transmitir mensagens entre aplicações ou organizações [37].

Para atender a requisitos de alta disponibilidade e escalabilidade, o *RabbitMQ* pode ser implementado em configurações distribuídas e federadas. Ele pode ser executado em diversos sistemas operativos e ambientes de nuvem, e suporta uma ampla variedade de linguagens de desenvolvimento e mecanismos de segurança, como autenticação, autorização, TLS e LDAP. As mensagens são encaminhadas através de *exchanges* antes de chegarem às *queues*. O *RabbitMQ* possui diversos tipos de *exchanges* integrados para lógica típica de roteamento. Mesmo em caso de falhas, este *broker* dispões de uma variedade de recursos para permitir que as mensagens sejam trocadas com confiabilidade, incluído persistência, garantias de entrega, confirmações de recebimento da mensagem, e alta disponibilidade através do espelhamento das *queue* [37]. Além disso, o *RabbitMQ* vem com uma variedade de *plugins* que o estendem de diferentes maneiras, sendo a comunidade de desenvolvimento bastante grande, produzindo diversos tipos de clientes, *plugins*, entre outros.

3.3.4. Resumo

Kafka, *ActiveMQ* e *RabbitMQ*, são todas tecnologias de mensagens utilizadas para fornecer comunicação assíncrona e desacoplar processos (separando o remetente e o destinatário de uma mensagem). Todos servem o mesmo propósito básico, mas podem realizar as suas funções de maneira diferente. Neste caso o *Apache Kafka*, pela sua escalabilidade, alto desempenho e alta capacidade de processamento de *streaming* de dados é aquele que está mais próximo daquilo que se pretende com a implementação deste sistema. A Tabela 3 descreve de forma resumida algumas diferenças e semelhanças entre essas plataformas.

	Kafka	*ActiveMQ*	*RabbitMQ*
Open-source	X	X	X
Suporte nativo a vários protocolos de mensagem		X	X
Suporta diversas linguagens	X	X	X
Persistência	X	X	X
Suporta mecanismos de segurança	X	X	X
Processamento de *Streaming* dados distribuídos	X		
Maior *throughput* de dados	X		
Solução Adotada	X		

Tabela 3 - Comparação entre o Kafka, ActiveMQ e RabbitMQ

3.4. Data Warehouse em tempo real

Em geral as implementações de *Data Warehouse* tradicionais utilizam o mesmo conceito do cubo de dados estático proposto por Gray et al. [38], e materializam vistas multidimensionais dos dados para garantir um desempenho elevado na execução das consultas. No entanto essa abordagem do cubo de dados estático tem várias desvantagens, pois os dados no cubo só podem ser atualizados periodicamente, portanto a última informação inserida no sistema OLTP, normalmente não é incluído no processo de consulta dos dados.

Diante disso, progressivamente têm surgido mais investigações e propostas de soluções de *Data Warehouse* em tempo real. Uma das abordagens utilizadas são sistemas de base de dados em memória, que implementam simultaneamente os sistemas OLTP e OLAP, como forma de diminuir drasticamente os tempos de resposta das consultas de dados. Depois com a evolução das bases de dados, surgiram os sistemas de armazenamento de séries temporais, que basicamente armazenam dados coletados em intervalos regulares durante um período de tempo, como forma de manter o histórico de eventos, e tornar o processamento de grandes quantidades de dados mais eficientes. Também o desenvolvimento das ferramentas de *Hadoop*, tem contribuído para o aparecimento de algumas implementações de *Data Warehouse* em tempo real, utilizando o modelo de programação *MapReduce* para processamento distribuído de grandes volumes de dados.

3.4.1. Hyrise

O *Hyrise* é um sistema híbrido de base de dados (OLTP e OLAP) de armazenamento sobretudo em memória, que divide tabelas em partições verticais de dimensões variadas, dependendo de quantas colunas da tabela são acedidas [39]. Para as colunas acedidas através de consultas OLAP, partições menores têm melhor desempenho, pois normalmente, essas consultas são aplicadas sobre um número reduzido de colunas. Enquanto nas consultas OLTP, frequentemente são feitas transações como *insert, update* ou *select* que normalmente exigem acesso a muitos campos de um registo, por isso partições maiores neste caso funcionam melhor. Para lidar com as partições criadas, o *Hyrise* utiliza um modelo altamente preciso de erros de cache, através do qual é capaz de prever o desempenho das diferentes partições, e com isso escolher a melhor partição para efetuar a pesquisa ou consulta dos dados requisitados [39].

3.4.2. Hyper

O *Hyper* é outro sistema de armazenamento capaz de lidar com ambos, OLTP e OLAP em simultâneo, utilizando mecanismos de replicação assistida por Hardware, para manter capturas instantâneas (*snapshots*) dos dados transacionais numa estrutura de base de dados em memória [40]. *Hyper* é sobretudo um sistema de base de dados em memória que garante a execução de transações OLTP, seguindo as regras das propriedades ACID, e a execução de múltiplas consultas OLAP sobre capturas instantâneas tiradas dos dados.

A arquitetura do *Hyper* foi criada para que as transações OLTP e OLAP não interfiram entre si. As transações OLTP são processadas a altas taxas por segundo, ao mesmo tempo que as consultas OLAP são executadas em capturas instantâneas e recentes dos dados. As consultas OLAP são somente de leitura, portanto podem facilmente serem executadas em paralelo através de múltiplas "*threads*" que partilham o mesmo espaço de endereçamento [40]. Isso evita sobrecarga ou bloqueio das consultas OLAP, sendo que elas não partilham entre si qualquer estrutura de dados mutável.

3.4.3. Druid

O *Druid* [41] é um sistema *open-source* de armazenamento distribuído e escalável, apropriado para análises em tempo real de grandes quantidades de dados. Foi

originalmente concebido para o consumo e exploração de eventos transacionais, e fornece escalabilidade para armazenar grandes volumes de dados de séries temporais. Este sistema de armazenamento é suportado principalmente por bases de dados em memória, mas por outro lado, é também considerado um sistema de armazenamento de séries temporais (*time series*), através do qual suporta múltiplas versões dos mesmos dados [42].

Tal como apresentado na Figura 5, a arquitetura do *Druid* é dividida em 4 nós principais (*Real Time*, *Coordinator*, *Historical* e *Broker*). À medida que os dados são inseridos, o *Druid* constrói e mantém um conjunto de índices em memória no nó em tempo real. A partir do momento em que os dados estiverem indexados, ficam logo disponíveis para consulta. Para evitar problemas de sobrecarga de memória, periodicamente ou após atingir o limite máximo, estes dados são transferidos para um mecanismo de armazenamento externo ao *Druid*, designado de *Deep Storage* (disco local ou partilhado, *hadoop* ou armazenamento compatível com S3). Sendo os dados posteriormente carregados do *Deep Storage* para o nó histórico para consultas, mediante instruções do nó coordenador [42]. Além disso, o *Druid* também permite carregamento de dados de forma estática, diretamente no *Deep Storage*, através de mecanismos de consumo em lote (*batch*).

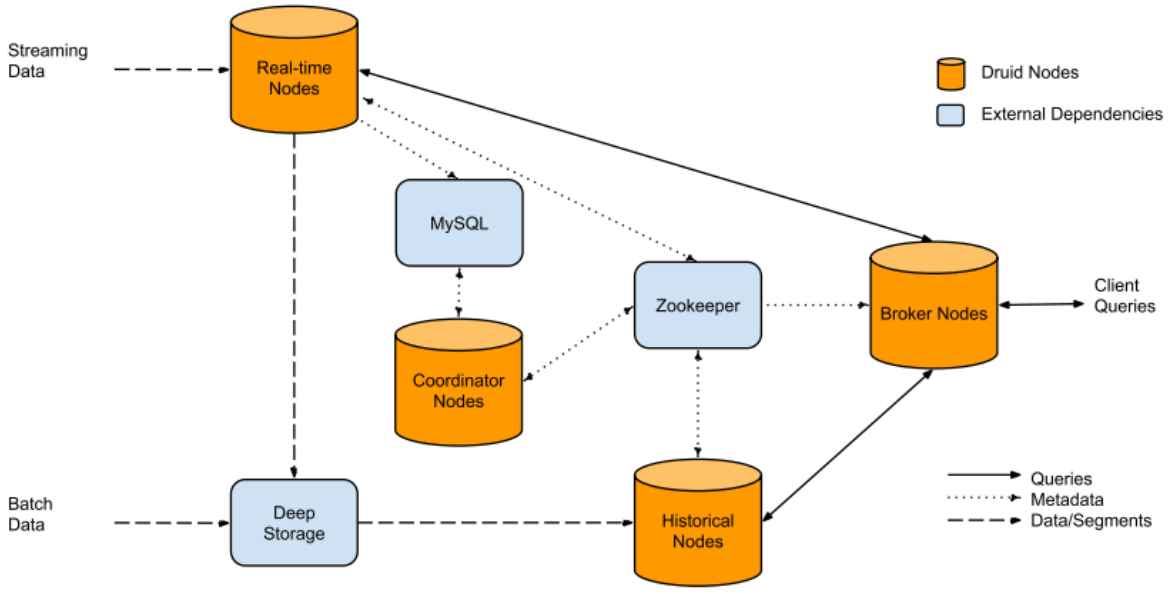

Figura 5 - Visão geral de um cluster Druid [42].

Para manter informações de configuração, nomeação, fornecer sincronização distribuída e serviços de grupo, normalmente é utilizado o *Zookeeper* [43]. A sua utilização evita que

diferentes implementações de serviços sejam criadas, e consequentemente facilita a implementação e a gestão desses mesmos serviços. Deste modo as informações de configuração são mantidas de forma centralizada [43].

3.4.4. R-Store

O *R-Store* é um sistema distribuído e escalável que utiliza o modelo de programação *MapReduce* para suporte a análises de dados em tempo real [44]. Esta solução utiliza o *HBase*, um armazenamento de dados distribuído, *open-source*, escalável, e orientada a coluna [45]. Pelo que, suporta múltiplas versões dos mesmos dados, e permite análises em tempo real, essencialmente devido a utilização da metodologia de programação *MapReduce* [46]. O Google desenvolveu o *MapReduce* como seu mecanismo para melhorar o sistema de indexação da internet e análise de *logs*. Um trabalho sobre este modelo de programação é apresentado no [47], onde é feito um estudo sobre a gestão de base de dados utilizando o *MapReduce*, e no [48] pode-se encontrar um estudo detalhado do desempenho deste *framework* num *cluster* com vários níveis de paralelismo.

Na arquitetura do *R-Store* apresentada na Figura 6, o *HBase* foi estendido como um sistema de armazenamento subjacente, que armazena tanto os dados do cubo, como os dados em tempo real. Este sistema de armazenamento é a base de dados que por padrão é utilizada no *Apache Hadoop*, o qual juntamente com seu sistema de ficheiro HDFS [31], foram tipicamente construídas tendo também como base o modelo de programação *MapReduce*. A plataforma *Hadoop* [49] é uma *framework* que permite o processamento distribuído de grandes conjuntos de dados em *clusters* de computadores. Ele foi projetado para escalar de servidores individuais para milhares de máquinas, cada uma oferecendo computação local e armazenamento [50].

Estes tipos de sistemas, pela forma como realizam um processamento distribuído dos dados, tornam-se indicados para serem utilizados em *cluster*, pelo que podem ser facilmente escalados à medida que as necessidades forem aumentando.

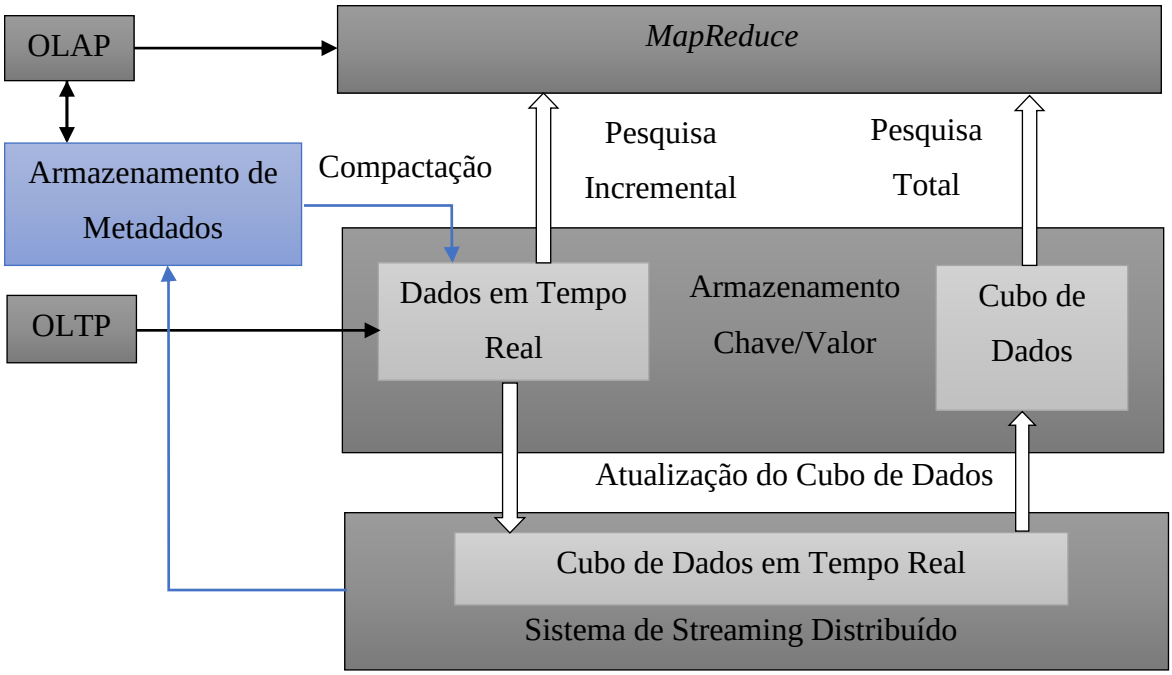

Figura 6 - Arquitetura do R-Store [44]

De acordo com a arquitetura do *R-Store*, as consultas OLTP são submetidas diretamente no armazenamento chave/valor, enquanto as consultas OLAP são primeiramente processadas pelo sistema *MapReduce*. Os dados em tempo real são verificados pela operação de pesquisa incremental, com base na última data em que o cubo de dados foi atualizado, enquanto no cubo de dados são efetuadas pesquisas completas. Quando os dados em tempo real são atualizados, eles são transmitidos para um sistema de *Streaming* distribuído, para que depois seja atualizado o cubo em uma base incremental. Com base nos metadados armazenados, o cubo de dados ou a base de dados OLTP, ou ambos, são utilizados pelas tarefas *MapReduce* para consultas OLAP. O *R-Store* dispõe de técnicas para pesquisas eficientes de dados em tempo real no sistema de armazenamento, e utiliza um algoritmo concebido para processar as consultas em tempo real, com base num modelo de custo proposto [44].

3.4.5. Resumo

Depois de analisadas as soluções de *Data Warehouse* anteriormente apresentadas, é necessário escolher uma delas. Essencialmente foram apresentados três tipos de soluções de implementação de *Data Warehouse* em tempo real, sistemas híbridos (OLTP e OLAP)

de armazenamento sobretudo em memória, base de dados de séries temporais, e soluções que utilizam *MapReduce* para o processamento dos dados.

Os sistemas híbridos como o Hyrise e o Hyper são sistemas de servidor único, com uma forma de funcionar muito dependente da memória disponível. Podem ser adequados para algumas aplicações, mas não são indicados para sistemas que processam fluxos de alta velocidade e consultas agregadas. Além disso implementar um sistema deste tipo, nesta fase implicaria alterar o sistema OLTP em produção, o que exigiria um esforço de implementação muito grande. Também as soluções de *Data Warehouse* que têm por base a metodologia de programação *MapReduce*, como o *R-Store*, não são os mais adequados para este cenário de implementação. Dado que o modelo de programação *MapReduce* requer que os programadores escrevam programas customizados os quais são difíceis de manter e reutilizar.

As bases de dados de séries temporais, pelo seu desempenho com grandes volumes de dados e pelas suas especificidades, também se adaptam aos requisitos de implementação deste *Data Warehouse*. Além do *Druid*, existem outras bases de dados de series temporais, com formas de funcionar bastante específicas e diferentes entre si, como o *Rhombus* [51], *InfluxDB* [52], *OpenTSDB* [53], *MonetDB* [54], *Blueflood* [55], *NewTS* [56] e o *KairosDB* [57]. Bader [58], [59] realizou comparações detalhadas, entre essas e outras bases de dados de series temporais bastante populares, mediante vários critérios previamente definidos, com intuito principal de determinar quais as bases de dados com melhor desempenho para cada um dos critérios definidos. Contudo de acordo com os resultados obtidos por Bader, o *Druid* é a melhor solução global se nenhum suporte comercial, nenhuma versão estável ou de suporte a longo prazo (LTS) for necessária. Assim, o *Druid* é a solução que melhor se adequa aos requisitos de funcionamento exigidos para este sistema, visto que em relação as outras bases de dados de series temporais, é aquele que tem melhor desempenho nos pontos considerados essenciais para o bom funcionamento deste *Data Warehouse*, como boa gestão do espaço de armazenamento e a capacidade com que consegue responder às consultas e agregações (contagens, somas e médias) no mais curto espaço de tempo. Apesar do *Druid* ainda ser uma solução bastante recente no mercado, e não ter nenhuma versão LTS, a comunidade de desenvolvimento desta ferramenta vem aumentando, e cada

vez é maior o número de empresas que utilizam esse sistema sobretudo como uma ferramenta analítica de dados.

3.5. Visualização de Dados

Visualização de dados é a representação gráfica de informações e dados. Permite aos decisores analisarem grandes quantidades de dados, através de elementos visuais como gráficos e mapas, com a finalidade de identificarem tendências, valores discrepantes e padrões nos dados [60]. A visualização interativa pode levar o conceito de interação com os dados um passo adiante, utilizando técnicas como *drill down* em gráficos e tabelas para obter mais detalhes, alterando interactivamente os dados visualizados.

Apesar do *Druid* suportar diversas ferramentas e tecnologias de visualização de dados, na documentação oficial é sugerido a utilização do *Airbnb*/*Superset* [61], *Grafana* [62], *Metabase* [63] ou Pivot [64]. No caso deste último é uma solução comercial e proprietária, pois é parte da plataforma analítica *Imply* [65], cuja distribuição utiliza o *Druid* como sistema de armazenamento de dados. Portanto, o Pivot não será considerado para este trabalho, visto que o mesmo se encontra fortemente vinculado a plataforma do *Imply*, o que dificulta ou mesmo impossibilita a sua utilização de forma independente [64].

3.5.1. Apache Superset

O *Superset* é uma plataforma *open-source*, de exploração e visualização de dados, originalmente desenvolvida pela *Airbnb* [66], projetado para ser visual, intuitiva e interativa [67]. Esta plataforma foi inicialmente chamado de *Panoramix*, depois foi renomeado para *Caravel* em março de 2016, e recentemente tornou-se um projeto da Apache [68], tendo sido denominado de *Superset*, desde novembro de 2016 [61]. Ele está configurado para suportar inúmeros tipos de visualizações, possibilitando aos utilizadores criar *dashboards* customizáveis. Foi construído numa *framework* extensível (*Flask*) [69], utilizando o *python*, que ultimamente tem-se tornado uma linguagem em clara ascensão na área da ciência dos dados.

O facto de ser uma ferramenta *open-source*, torna a sua utilização bastante atraente, sendo que os utilizadores podem ter acesso não apenas a painéis analíticos, mas também a

ferramentas poderosas para fazer variadíssimos tipos de análises. Embora se possa utilizar o *Superset* em um ambiente de teste numa configuração modesta, praticamente não há limites de expansão desta plataforma, visto que sendo uma ferramenta *web*, é compatível com a maioria dos navegadores de internet. Também é flexível, na medida em que permite a escolha do servidor *web*, da base de dados de metadados, do *broker* de mensagens, entre outras facilidades [61]. Um dos desafios lançados pela *Airbnb* é para que o acesso a dados através desta ferramenta, seja para todos os utilizadores, independentemente do nível de conhecimento que o mesmo possa ter sobre base de dados, a fim de tomarem decisões informadas. Os utilizadores podem visualizar os dados armazenados, provenientes de uma variedade de sistemas de armazenamento, inclusive o *SQL Server* que é a base de dados utilizado pelo SIJ. Além disso, é possível conectar esta ferramenta de visualização ao *Druid*, com o intuito principal de visualizar grandes quantidades de informações [67].

3.5.2. Grafana

O *Grafana* é também uma ferramenta de visualização *open-source* para análise de dados. É utilizado para uma variedade de casos, incluindo *DevOps*, *IoT*, e *AdTech* [70]. Oficialmente esta ferramenta também suporta diferentes tipos de fontes de dados. Para cada fonte de dados existe um editor de consultas de dados customizado para as características e capacidades da base de dados utilizada. Essencialmente, o *Grafana* é um substituto rico em recursos para o *Graphite-web*, que ajuda os utilizadores a criar e editar facilmente painéis de visualização de dados. Os utilizadores podem criar vários tipos de gráficos, como resultado da rápida renderização do lado cliente do *Grafana*. Adicionalmente, o *Grafana* possui um mecanismo de alerta integrado que permite aos utilizadores incluir regras no painel de controlo que despoletam eventos e consequentemente alertas para um *endpoint* de notificação [70].

A comunidade de contribuidores e desenvolvedores de *plugins* do *Grafana* é bastante numerosa e ativa. Atualmente existem vários *plugins* sendo desenvolvidos e constantemente melhorados com os mais diversos propósitos, entre os quais um *plugin* para estabelecer ligações de dados com o *Druid*.

3.5.3. Metabase

O *Metabase* é uma simples e poderosa ferramenta *open-source*, projetada para facilitar e agilizar o processo de análise de dados. Tal como as ferramentas de visualização apresentadas nos tópicos anteriores, o *Metabase* suporta ligações a vários tipos de base de dados. É possível executar o *Metabase* em qualquer máquina, independentemente do sistema operativo, desde que tenha o *Java Virtual Machine* instalada. A forma mais simples de o colocar a funcionar, é executando o ficheiro *jar* (*metabase.jar*), disponibilizado sem qualquer custo monetário no repositório oficial do *Metabase* [71]. Esta ferramenta de visualização permite que qualquer utilizador, mesmo aqueles que não detêm conhecimentos técnicos, possam tomar decisões com base em questões (*queries*) feitas sobre os dados armazenados. As *queries* feitas as bases de dados podem ser guardadas para serem utilizadas posteriormente, facilitando a sua reutilização, ou então para que os seus resultados sejam agrupadas em painéis na *dashboard* através de gráficos ou tabelas.

O construtor de *queries* do *Metabase* apenas permite executar consultas simples sobre uma única tabela na base de dados. Apesar de ser possível extrair informações de tabelas vinculadas, e modificar colunas para criar novas colunas com base em expressões matemáticas, o *Metabase* foi projetado tendo como principal requisito a usabilidade. Esta ferramenta foi desenvolvida com o foco principal nas pessoas que não têm qualquer conhecimento técnico, sobretudo a nível do *SQL* [72]. Por intermédio do construtor de *queries* não é possível fazer junções de tabelas, mas para este caso pode-se utilizar o editor de *queries* do *Metabase*, sendo que isto exige que o utilizador tenha conhecimentos necessários, como por exemplo em SQL para fazer o agrupamento de tabelas.

3.5.4. Resumo

As ferramentas de visualização apresentadas registam diferenças entre eles no que diz respeito à sua estrutura. No entanto, estas diferenças são mínimas, principalmente entre *Apache Superset* e *Metabase*, tendo em consideração as funcionalidades que se pretende incorporar neste trabalho. Na Tabela 4 é apresentada o resumo das características anteriormente especificadas e destaca a solução adotada.

	Superset	*Grafana*	*Metabase*
Integração com o *Druid*	X	X (*plugin*)	X
Facilidade de Utilização			X
Maior número de tipos de visualizações	X		
Editor de consultas nativas do *Druid* (Json)			X
Melhor integração com o *Druid*	X		X
Solução Adotada			X

Tabela 4 - Comparação entre o Apache Superset, Grafana e Metabase.

3.6. Considerações finais

Neste capítulo foram apresentados os fundamentos teóricos utilizados para a implementação e interligação de todos os componentes de um sistema de *Data Warehouse* em tempo real. Neste sentido, foram abordadas um conjunto de ferramentas e métodos que podem ser utilizados para a deteção, extração, transformação, carregamento, distribuição, armazenamento e apresentação dos dados. Cada uma dessas fases irá equivaler a um componente da arquitetura que se pretende implementar, no qual serão utilizados apenas as soluções adotadas.

4. Desenho e Arquitetura

A arquitetura de um sistema consiste essencialmente na definição dos componentes, suas propriedades externas, e seus relacionamentos com outros sistemas. Portanto, neste capítulo serão descritos os componentes da arquitetura do *Data Warehouse* em tempo real, e os relacionamentos que estes mantêm entre si e com outros sistemas.

4.1. Visão Geral

Em todo caso, para não afetar o desempenho do SIJ, a arquitetura deste sistema de *Data Warehouse* em tempo real foi projetada para ser implementada separado do ambiente de produção, sendo constituída por componentes como a captura de dados alterados (CDC), a extração, transformação e carregamento dos dados (ETL), um *broker* de mensagens, um sistema de armazenamento para grandes quantidades de dados (DW), uma *interface* de acesso aos dados armazenados no *Data Warehouse* (DW Manager), e uma ferramenta de visualização de dados (Figura 7).

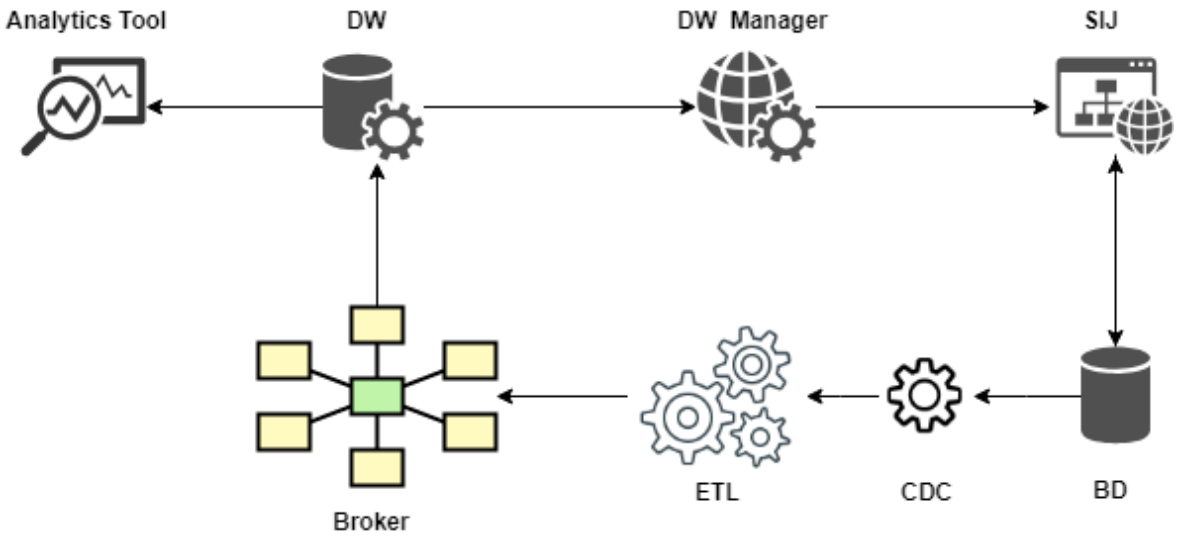

Figura 7 - Arquitetura do Data Warehouse em tempo real para o SIJ

Os dados do SIJ são armazenados essencialmente numa única base de dados, designada de MJCVProcessos, o qual é constituída por inúmeras tabelas que representam a estrutura de dados onde são armazenados as informações relativos aos processos dos tribunais, e seus intervenientes.

Quando uma operação do tipo *insert*, *update* ou *delete* for feita a base de dados do SIJ, essas alterações são registadas no *log* da respetiva base de dados. Com o CDC habilitado, é possível ler estes dados registados no *log*, e armazena-los juntamente com metadados em tabelas que espelham as propriedades e atributos das tabelas de origem. Isto possibilitou a utilização de uma ferramenta ETL, capaz de extrair apenas os dados mais recentes ou as últimas alterações dos dados, efetuar as transformações necessárias sobre os dados, e posteriormente envia-los para o *broker*. Os dados no *broker* são organizados por tópicos ou assuntos, e podem ser consumidos mediante a subscrição de um determinado tópico. Efetivamente a principal característica do *broker*, para além de único ponto de convergência e segurança dos dados, é a escalabilidade, o que permite aumentar o número de componentes na arquitetura, conforme as necessidades para atender aos requisitos de desempenho. À medida que os dados são inseridos no *broker*, vão sendo consumidos e armazenados no *Data Warehouse*. Este por sua vez, através de mecanismos como indexação, agregações e pré-cálculos, permite que as consultas aos dados armazenados sejam efetuadas no menor tempo possível.

Entretanto, foi preciso implementar uma forma para que o SIJ consiga aceder aos dados armazenados. Desta forma, foi criado o DW Manager, o qual fornece uma *interface* programática para acesso aos dados armazenados no *Data Warehouse*. O DW Manager deverá fornecer compatibilidade entre esses dois sistemas, permitindo ao SIJ interagir com os dados armazenados independentemente da linguagem de consulta suportada pelo *Data Warehouse*.

Além de tudo o que já foi mencionado, esta arquitetura integra uma ferramenta de visualização e análise dos dados (*Analitics Tool*). Esta ferramenta torna-se extramente importante sobretudo para os decisores ou outros órgãos responsáveis pelas tomadas de decisões, na medida em que possibilita a análise dos dados em tempo real, através de gráficos e outras formas de visualização, permitindo tomadas de decisões de forma mais atempada e consequentemente mais eficazes.

4.2. Extração, transformação e carregamento dos dados

ETL é um componente fundamental na arquitetura para o funcionamento do *Data Warehouse* em tempo real. Pois, quanto menor for o intervalo de tempo, entre fazer uma transação (*insert*, *update* e *delete*) na base de dados, e o momento em que essa transação é detetado pelo processo ETL, mais rapidamente é despoletado o processo de atualização dos dados no *Data Warehouse*. Para isso, o ETL teve de ser configurado para ser executado periodicamente em intervalos mais próximos uns dos outros, ou seja, no momento em que a transação é detetada na base de dados do SIJ.

Na Figura 8, é ilustrado o diagrama de fluxo de operações ETL dos dados do SIJ, desde a deteção de novas transações nas tabelas CDC, os quais refletem a mesma estrutura de dados das tabelas de origem, até o carregamento dos dados no *broker*. Cada operação neste fluxo é executada sequencialmente e validada antes de executar a operação seguinte. Isto evita a execução de operações com erros, e permite uma melhor depuração dos erros que possam ocorrer.

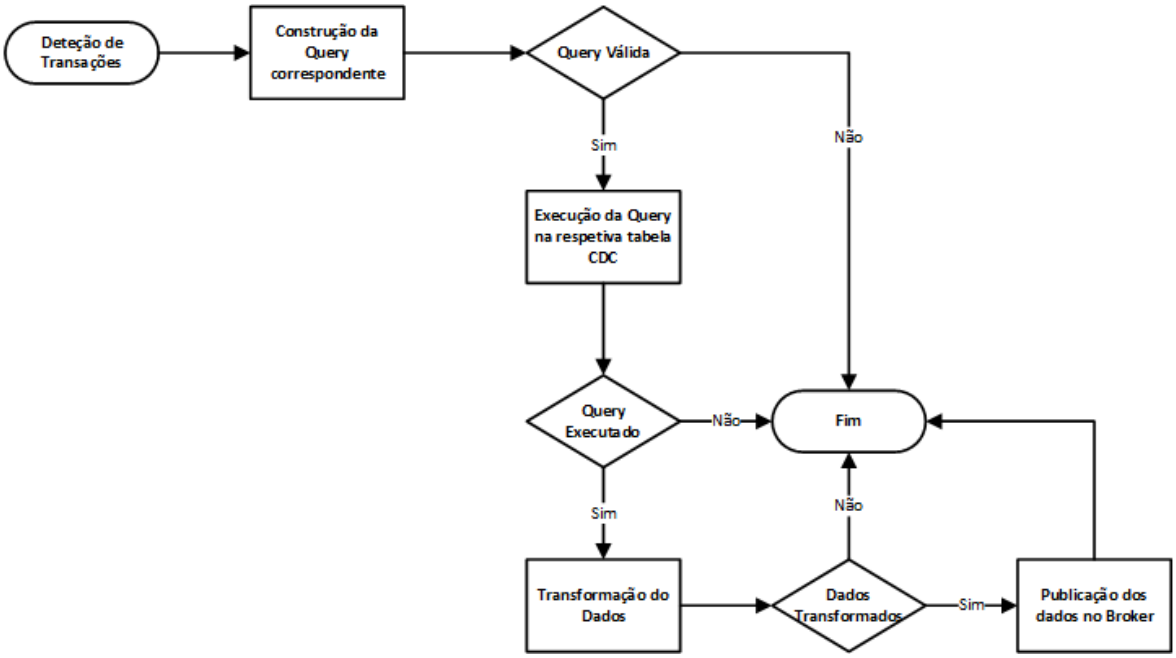

Figura 8 - Diagrama do fluxo de dados do processo ETL

Adicionalmente, para além destas ferramentas ETL serem compatíveis com vários sistemas de base de dados, também permitem efetuar várias transformações sobre os dados, como por exemplo conversões de formato de dados.

4.3. Consumo de dados em tempo real

Um dos grandes desafios deste trabalho foi o consumo dos dados em tempo real por parte do *Data Warehouse*. Numa primeira abordagem foi utilizado o *Tranquility*, o qual pode ser definido como um projeto de um servidor que é executado de forma independente, e que envia *streams* de dados para o *Druid*. Contudo, o *Tranquility* tem algumas limitações, pois é necessário configurar a janela temporal dos dados, onde apenas os dados gerados num intervalo de tempo dentro dessa janela temporal são consumidos pelo *Tranquility*. Evidentemente que este problema poderia ser em parte resolvido, aumentando a janela temporal de consumo dos dados em tempo real. Em parte, porque ainda assim o *Tranquility* continuava a falhar no consumo de alguns dados, neste caso devido a erros de incompatibilidade quanto ao formato de dados de um dos campos do tipo *datetime* utilizado no processo de indexação. Em concreto, uma simples alteração no formato de dados deste campo em ordem ao tempo, para que o registo não fosse consumido pelo

processo de indexação do *Tranquility*, pelo que perante este cenário, esta não é a solução ideal para consumo dos dados em tempo real. Por isso, numa segunda abordagem foi utilizado o serviço de indexação *Kafka*, que tem por base o mesmo princípio de funcionamento do serviço de indexação do *Druid*, sendo definido como um serviço distribuído e altamente disponível, que executa tarefas relacionadas a indexação. O serviço de indexação tem uma arquitetura cliente/servidor, composta pelo *peon* que pode executar uma única tarefa, o *Middle Manager* que faz a gestão dos *peons*, e o *overlord* que efetua a gestão da distribuição de tarefas para o *Middle Manager* (Figura 9) [73].

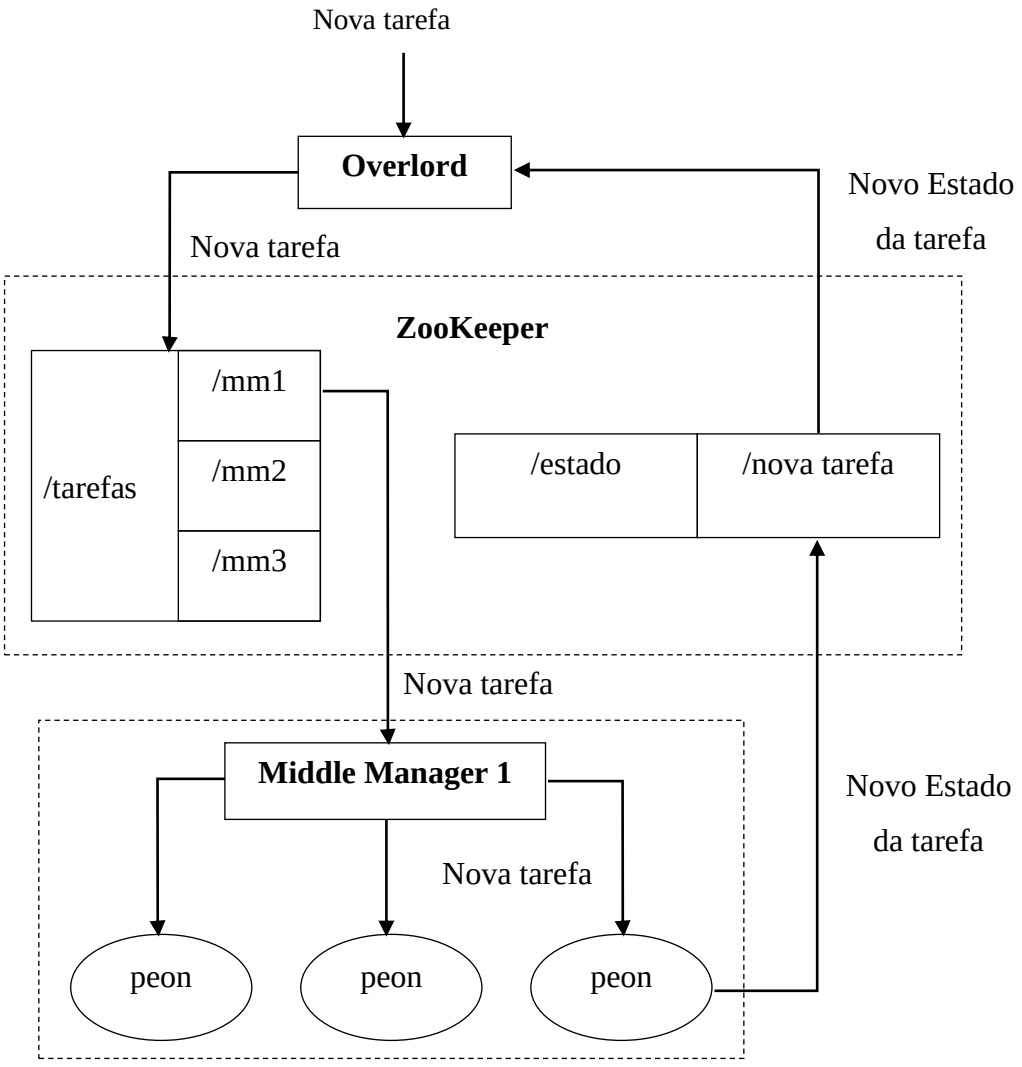

Figura 9 - Arquitetura do serviço de indexação [73]

Apesar do serviço de indexação *Kafka* do *Druid* estar ainda numa fase experimental, tem tido cada vez mais apoiantes no sentido de tornar-se o padrão quando utilizado juntamente com o *Kafka*, pelo que foi a solução utilizada. Dado que ele permite a configuração de supervisores no *Overlord*, que facilitam o consumo de dados a partir do *Kafka*, gerindo a criação e o tempo de vida das tarefas de indexação. Essas tarefas de indexação leem eventos utilizando o próprio mecanismo de partição e *offset* (deslocamento) do *Kafka* e, portanto, são capazes de fornecer garantias de ingestão de dados exatamente uma vez. Isto é conseguido porque o serviço de indexação mantém o registo do último *offset* persistido no *Kafka* por forma a garantir que a mesma mensagem seja consumida apenas uma vez entre as tarefas. As tarefas subsequentes devem começar a ler a partir de onde a tarefa anterior foi concluída, para que os segmentos criados sejam aceitos. Também o serviço de indexação é capaz de ler eventos não recentes de *Kafka*, e não está sujeito às considerações de período de janela impostas pelos outros mecanismos de consumo de dados em tempo real. O supervisor supervisiona o estado das tarefas de indexação para coordenar *handoffs*, gerenciar falhas e garantir que os requisitos de escalabilidade e replicação sejam mantidos [74].

4.4. Interface de consulta de dados do Data Warehouse

De modo a simplificar o acesso aos dados armazenados no *Data Warehouse*, foi construída uma *interface*, mais especificamente o DW Manager que esconde toda a complexidade das *queries* nativas do *Data Warehouse*. Para aceder diretamente aos dados armazenados no *Data Warehouse* é preciso utilizar um tipo de *query* diferente do padrão SQL, com uma estrutura própria. Assim, o DW Manager disponibiliza uma API que encapsula chamadas, que dependendo do tipo de *query*, recebe um número variável de parâmetros de entrada, necessários para a execução dessas *queries*.

O DW Manager, tal como ilustrado na Figura 10, é constituído essencialmente pelo *Query Models*, *Post Query*, *SIJDW Manager*, *SIJ Controller* e *Response Model*. Ele permite quatro tipos de *queries* suportadas pelo *Data Warehouse*: *Select*, *Time Series*, *TopN* e *GroupBy*, sendo que ambos estendem o *Query Models*.

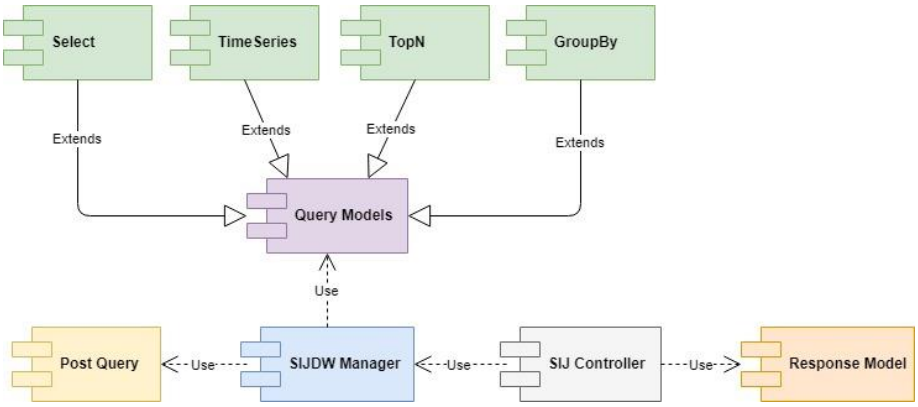

Figura 10 - Diagrama de componentes do DW Manager

O *SIJDW Manager* é utilizado para construir a estrutura de dados das *queries*, através do *Query Models*, podendo ser um dos quatro tipos de *queries* anteriormente mencionados. Com a estrutura de dados da *query* definida, utiliza-se o *Post Query* para converter essa estrutura no formato de dados suportado pelo *Data Warehouse*, o qual é de seguida submetida através de um URL especificamente concebido para tal. Para finalizar, o *SIJController* obtém a resposta da *query* efetuada, através do *SIJDW Manager*, e utiliza o *Response Model* para converter a resposta no formato desejado.

Estes componentes podem ser facilmente transformados em serviços ou micro serviços, de forma que possam ser utilizados por diversos outros sistemas para além do SIJ, com basicamente as mesmas necessidades.

5. Implementação

Este capítulo foca-se em detalhes de implementação da arquitetura do *Data Warehouse* em tempo real. Tendo como base o diagrama apresentado na Figura 11, com indicações das tecnologias e sistemas utilizados na arquitetura desenvolvida, serão descritos os detalhes e as configurações envolvidas no processo de implementação.

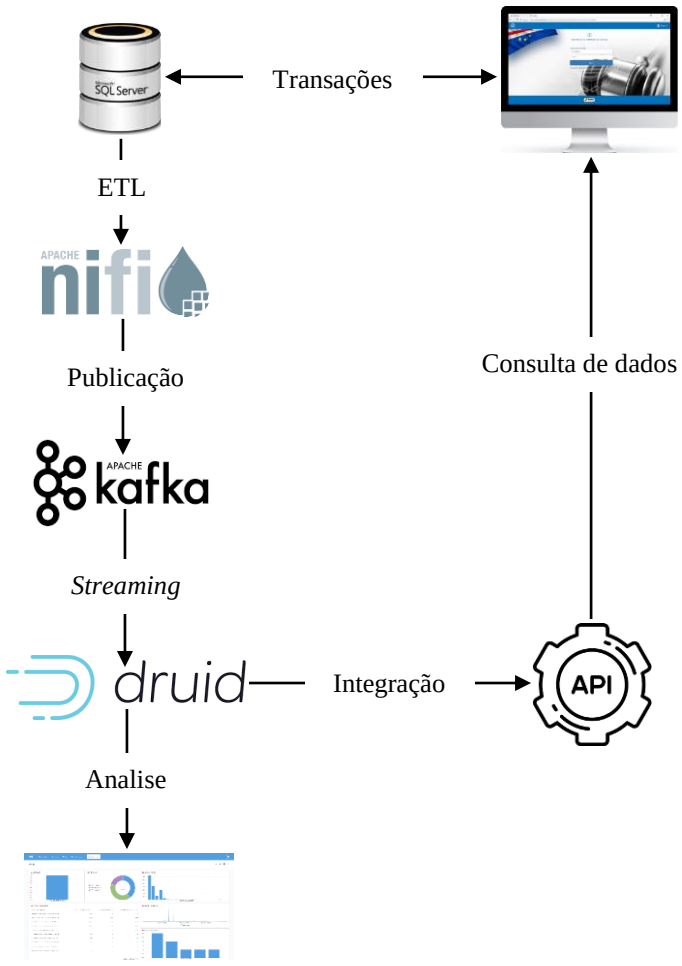

Figura 11 - Cenário de Implementação da arquitetura desenvolvida

Primeiramente será descrito como ativar e configurar o CDC do *SQL Server*, através do qual é possível monitorizar as tabelas da base de dados do SIJ, considerados relevantes para a implementação do *Data Warehouse* em tempo real. Dando seguimento a este processo de captura e transformação dos dados, será explicado o processo de configuração do *Apache Nifi*, utilizado como ferramenta de extração, transformação e carregamento dos dados no *Data Warehouse*, onde a transferência de dados entre o *Apache Nifi* e o *Data Warehouse* é feita com o auxílio de um *broker* de mensagens, mais especificamente o *Apache Kafka*. De seguida, será detalhado todo o processo de configuração do *Druid*, através do qual efetua-se a leitura dos dados armazenados no *Kafka* utilizando serviços de indexação, sendo posteriormente armazenados e disponibilizados para consulta. Depois, para permitir que estas consultas de dados fossem feitas a partir do SIJ, foi implementada o DW Manager, o qual também será abordado neste capítulo de forma mais pormenorizada. Além disso será apresentado os detalhes de configuração do Metabase, utilizado como uma ferramenta de análise e apoio a decisão.

5.1. Preparação da estrutura de dados

Antes de ativar o CDC, e para ter um maior controlo das versões dos registos armazenados, foi criado nas tabelas de origem dos dados um campo do tipo *datetime*, com o intuito principal de permitir identificar as últimas alterações efetuadas aos dados. Na Figura 12 é apresentado um conjunto de instruções *SQL* utilizadas para adicionar o campo *DataAccao* nas tabelas Auto, Despacho e Requerimento. A necessidade de adicionar este campo, justifica-se pelo facto dessas referidas tabelas não possuírem diretamente na sua estrutura, nenhum atributo que permita registar a data e hora em que as alterações foram feitas, tanto a nível de transações do tipo *insert* como de *update*.

```
USE MJCVProcessos
ALTER TABLE dbo.Auto ADD DataAccao datetime NOT NULL DEFAULT getdate()
ALTER TABLE dbo.Despacho ADD DataAccao datetime NOT NULL DEFAULT getdate()
ALTER TABLE dbo.Requerimento ADD DataAccao datetime NOT NULL DEFAULT getdate()
ALTER TABLE dbo.Requerimento ADD Valido bit NOT NULL DEFAULT 0
```

Figura 12 - Script utilizado para adicionar um campo do tipo datetime

O script apresentado na Figura 12, funciona perfeitamente no caso de transações do tipo *insert*, já que no momento em que o campo é inserido, este é definido automaticamente

com a data atual. Mas para transações do tipo *update* ainda é preciso definir o comportamento deste campo. A forma talvez mais óbvia é através do SIJ, o que implicaria fazer modificações nas transações do tipo *update* realizadas através da aplicação do SIJ, no entanto pretende-se efetuar o mínimo de alterações possível na parte lógica do sistema. Outra solução, é a utilização de *triggers* que atuam diretamente sobre a base de dados. Neste caso, os *triggers* foram a solução adotada porque com este método, as alterações são efetuadas apenas ao nível da base de dados.

Na Figura 13 é apresentado um exemplo de um *trigger*, cuja função é de monitorizar as alterações no campo DataDescricao, o qual provoca um processo em cascata para atualizar o campo DataAcao com a data atual. Deste modo, consegue-se detetar quando um auto, despacho ou requerimento são explicados com base no preenchimento da DataDescricao, sendo atualizado automaticamente a data em que a operação foi realizada.

```
CREATE TRIGGER [dbo].[trg_updateAutoDataAccao]
   ON   [dbo].[Auto]
   AFTER update
AS
BEGIN
    -- SET NOCOUNT ON added to prevent extra result sets from
    -- interfering with SELECT statements.
    SET NOCOUNT ON;

    -- Insert statements for trigger here
    IF UPDATE (DataDescricao)
    BEGIN
        UPDATE Auto
        SET DataAccao = GETDATE()
        FROM Auto A inner join Inserted I
        ON A.AutoID = I.AutoID
        WHERE I.DataDescricao is not null
    END
END
```

Figura 13 - Trigger para atualizar a DataAccao da tabela Auto

5.2. Configuração do CDC SQL Server para a base de dados do SIJ

O CDC armazena em tabelas especificamente criadas para esse efeito, todas as transações de manipulação de dados (*insert*, *update* e *delete*) aplicadas as tabelas monitoradas. Considerando a importância de manter os dados principalmente para auditoria, a aplicação

web do SIJ é contemplada apenas com operações do tipo *insert* e *update*, pelo que nas tabelas criadas através do CDC estarão refletidas apenas estes dois tipos de transações.

A tarefa de captura dos dados (*Capture Job*) é iniciada executando a *stored procedure* sp_MScdc_capture_job, o qual é criado no momento em que o CDC é ativado. Neste caso em específico, o CDC foi ativado conforme o exemplo apresentado na Figura 14. Esta *stored* a partir do momento em que é iniciada, começa pela extração dos valores configurados para *continuous*, *maxtrans*, *maxscans* e *pollinginterval* para a tarefa de captura especificada em msdb.dbo.cdc_jobs. Esses valores configurados são então transferidos como parâmetros para o procedimento sp_cdc_scan que por sua vez é utilizado para invocar o procedimento sp_replcmds, responsável pelo scan dos dados no log de transações.

```
USE MJCVProcessos
EXEC sp_changedbowner 'mjcvdeveloper'
EXEC sys.sp_cdc_enable_db
```

Figura 14 - Script para ativar o CDC no MJCVProcessos

Existem quatro parâmetros utilizados na *stored* sp_cdc_scan que determinam o comportamento da tarefa de captura de dados. O parâmetro designado de *continuous* determina quando as tarefas de captura de dados são executadas continuamente, ou quando terminam após uma única varredura a base de dados. Enquanto os outros parâmetros determinam com que regularidade ou quantas transações são lidas do log e inseridas nas tabelas do CDC. Por exemplo, o parâmetro *maxtrans* determina quantas transações são lidas do log em cada ciclo de leitura, sendo que os ciclos de leitura são definidos através do parâmetro *maxscan*. Após atingir o valor máximo definido no *maxscan*, as tarefas de captura de dados são colocadas em pausa, isto caso o parâmetro *continuous* for definido com o valor 1, caso contrário as tarefas de captura são finalizadas. Em que, a duração do intervalo da pausa é definida através do parâmetro *pollinginterval*. Na hipótese de o log não ter mais transações para serem lidas, ou se os ciclos de leituras estiverem completos, então é executado o WAITFOR.

Os parâmetros da tarefa de captura de dados foram configurados utilizando o procedimento sys.sp_cdc_change_job, conforme ilustrado abaixo na Figura 15.

```
EXECUTE sys.sp_cdc_change_job
    @job_type = N'capture',
    @maxtrans = 500,
    @maxscans = 10,
    @pollinginterval = 5;
```

Figura 15 - Alteração dos parâmetros da tarefa de captura de dados

O CDC utiliza uma estratégia de limpeza baseada no valor definido na retenção dos dados, para fazer a gestão do tamanho das tabelas que armazenam as alterações. Uma única tarefa de limpeza, quando executada é aplicada a todas as tabelas que armazenam as alterações monitoradas pelo CDC na base de dados, e aplica o mesmo valor de retenção a todas as instâncias de captura definidas.

A tarefa de limpeza (*Cleanup Job*), é iniciada executando a *stored procedure* sp_MScdc_cleanup_job, também criado no momento em que o CDC foi ativado. Ao executar esta *stored procedure*, primeiramente os valores de retenção e *threshold* configurados são extraídos da tarefa de limpeza do msdb.dbo.cdc_jobs. O valor de retenção específica em minutos quanto tempo os registos são mantidos nas tabelas de alterações do CDC. O valor padrão para a retenção dos dados é de 4320 minutos ou seja 72 horas/3dias. Enquanto o parâmetro *threshold* é utilizado para limitar o número de registos ou tuplas que podem ser removidas. O valor padrão para o *threshold* é de 5000 registos. Os valores desses parâmetros devem ser ajustados tendo em consideração o quão rápido as tabelas de alterações crescem em termos de dados. Na Figura 16, é apresentado um exemplo da utilização do procedimento sp_cdc_change_job, para parametrizar a tarefa de limpeza dos dados nas tabelas de alterações do CDC.

```
EXEC sp_cdc_change_job
@job_type='cleanup',
@retention = 4320,
@threshold = 5000
```

Figura 16 - Alteração dos parâmetros do cleanup job

Depois de configuradas as tarefas de captura e limpeza de dados do CDC, foram definidos quais as tabelas e os respetivos atributos que serão monitorados. Na Figura 17, é apresentado o script utilizado para configurar o monitoramento da tabela Auto, e capturar os valores dos campos definidos na lista de colunas capturadas.

```
EXEC sys.sp_cdc_enable_table
    @source_schema = N'dbo'
  , @source_name = N'Auto'
  , @role_name = NULL
  , @capture_instance = NULL
  , @supports_net_changes = 1
  , @captured_column_list = N'AutoID, ProcessoID, QuemSubmeteAuto, DataEntradaSistema,
       DataLimiteApreciacaoAuto, DataDescricao, DataAccao, DescritorDoAuto, Terminado'
  , @filegroup_name = N'PRIMARY';
GO
```

Figura 17 - Script utilizado para monitorizar uma tabela

Com o CDC funcionando segundo os parâmetros definidos, sempre que houver qualquer transação de manipulação de dados (DML) nas tabelas monitoradas, essas transações são refletidas em tabelas criadas pelo CDC, que espelham a mesma estrutura de dados das tabelas de origem correspondentes.

5.3. Preenchimento das tabelas CDC

Desde a data em que a base de dados do SIJ começou a ser utilizada ainda numa fase de testes, tem vindo a acumular nas suas tabelas um número considerável de registos. Como estes dados foram inseridos numa fase anterior a ativação do CDC, logo não foram considerados pelo mecanismo de captura. Neste caso a solução implementada para inserção dos respetivos dados nas tabelas CDC, foi simular transações do tipo *update* para todos os registos armazenados nas tabelas da base de dados do SIJ, atualizando cada coluna com exatamente os mesmos valores, tal como o exemplo ilustrado na Figura 18.

```
UPDATE dbo.auto
   SET QuemSubmeteAuto = QuemSubmeteAuto
      ,ProcessoID = ProcessoID
      ,DataEntradaSistema = DataEntradaSistema
      ,Terminado = Terminado
      ,DescritorDoAuto = DescritorDoAuto
      ,DataLimiteApreciacaoAuto = DataLimiteApreciacaoAuto
      ,DataDescricao = DataDescricao
      ,DataAccao = GETDATE()
```

Figura 18 - Instrução SQL para simular uma atualização em massa

Cada registo atualizado nas tabelas de origem, equivale a inserção de pelo menos dois registos nas respetivas tabelas CDC, um representando os valores anteriores a atualização (com o código de operação 3), e o outro com os valores posteriores (com o código de

operação 4), pelo que foram construídas instruções SQL, para eliminar os registos duplicados. Assim sendo, na Figura 19, é apresentado uma sequência de instruções SQL, utilizados em primeira instancia para atualizar alguns campos específicos e depois para eliminar todos os registos duplicados.

```
UPDATE cdc.dbo_Auto_CT
SET DescritorDoAuto = NULL
   ,DataDescricao = NULL
   ,DataAccao = GETDATE()
WHERE __$operation = 3

;With CTE_Duplicates as
(select cdc.dbo_Auto_CT.* ,
row_number() over(partition by cdc.dbo_Auto_CT.AutoID
order by cdc.dbo_Auto_CT.AutoID) rownumber
from cdc.dbo_Auto_CT)

delete from CTE_Duplicates where rownumber>1 and rownumber<4
```

Figura 19 - Script utilizado para eliminar registos duplicados

Terminado o processo de preenchimento e remoção de duplicados nas tabelas criadas pelo CDC, ao efetuar a contagem do número de registos em cada uma dessas tabelas, obteve-se uma quantidade igual ou superior aos das tabelas de origens (Tabela 5). Nos casos em que o número de registos nas tabelas CDC são maiores do que nas tabelas de origem, essas diferenças estão diretamente relacionadas com a quantidade de tarefas pendentes dos utilizadores do SIJ, no que se refere aos autos, despachos e requerimentos. De forma mais detalhada, cada uma dessas tabelas contém na sua estrutura um campo referente ao autor da explicação do auto, despacho ou requerimento dependendo do caso. Se este campo relativo ao autor estiver preenchido, significa que o cumprimento deste auto, despacho ou requerimento já foi efetuado, e esta tarefa já não se encontra mais pendente, caso contrário este campo é colocado com o valor NULL. Neste sentido, o valor da diferença apresentada na Tabela 5 para cada fonte de dados, corresponde à quantidade de autos, despachos ou requerimentos que já foram cumpridos ou explicados de entre o total de registos nas tabelas de origem. Dado que, estes tipos de registos são armazenados duas vezes em cada tabela criadas pelo CDC (antes e depois da explicação), onde a diferença entre os pares de registos esta apenas no campo referente ao autor da explicação.

Número de Registos	Processo	Auto	Despacho	Requerimento
Tabelas de Origem	124799	3	20614	8
Tabelas CDC	124799	5	31884	8
Diferença	0	2	11270	0

Tabela 5 - Comparação entre as tabelas de origem e respetivas tabelas CDC

Em relação ao auto do total de três registos, dois foram cumpridos, portanto desses três, dois estão em duplicado na tabela CDC, diferenciando-se apenas no campo referente ao autor da explicação. O mesmo acontece com o despacho. Enquanto com o requerimento a diferença é zero porque nenhum desses requerimentos foram cumpridos ou explicados.

5.4. Configuração do Apache Nifi como ferramenta ETL

A utilização do *Apache Nifi* como ferramenta de extração, transformação e carregamento dos dados do SIJ, deve-se sobretudo pelo conjunto de recursos que esta ferramenta disponibiliza, e pela facilidade com que permite projetar e configurar fluxos de dados, utilizando uma interface visual de arrastar e soltar bastante intuitiva.

No *Apache Nifi* foi configurado um conjunto de processos, cuja finalidade primeiramente é de extrair os dados das tabelas CDC, conforme estes forem sendo inseridos, efetuar o devido tratamento desses dados, e posteriormente encaminhá-los para o seu destino. Cada processo é normalmente atribuído a um processador diferente, utilizados normalmente para "escutar" os dados recebidos; extrair dados de fontes externas; rotear, transformar, e retirar informações do fluxo de dados. Em todas as conexões entre os processadores do fluxo de dados, existe uma *queue*, cuja finalidade é de garantir a entrega dos dados no processador de destino, mediante uma ou várias tentativas de entrega. A medida que os dados passam pelos processadores, cada um deles efetua um processamento específico sobre os mesmos, e depois reencaminha-os para o processador pelo qual está conectado. Neste contexto, encontra-se representado na Figura 20, um grupo de processadores interligados entre si, com a finalidade de extrair, transformar, e por fim publicar os dados no Apache Kafka. Apesar do exemplo apresentado ser referente aos autos, com ligeiras alterações este fluxo também se aplica aos dados provenientes dos despachos e requerimentos.

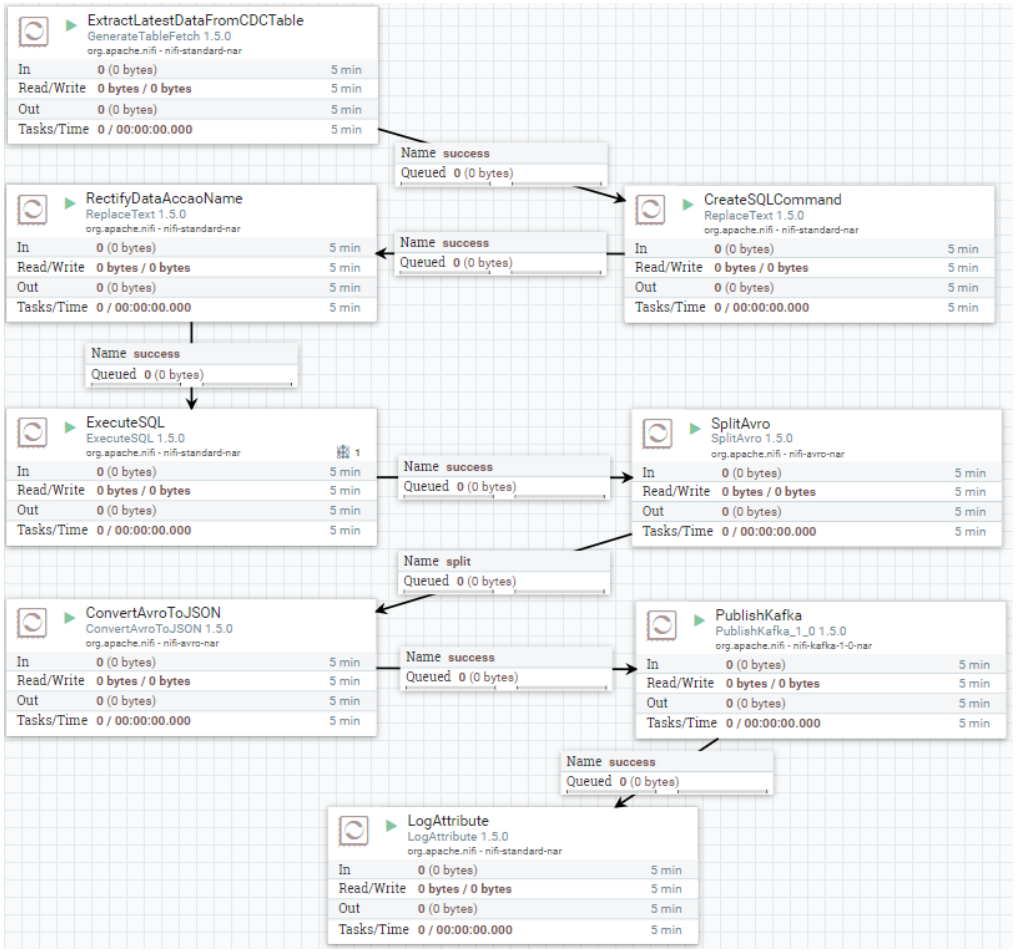

Figura 20 - Fluxo de dados ETL construído no Apache Nifi

O primeiro processador de dados do fluxo, designado de *ExtractLatestDataFromCDCTable*, é utilizado para gerar uma simples instrução SQL, capaz de extrair apenas os dados mais recentes, com base na data em que estes foram inseridos ou alterados. Esta instrução é depois alterada no processador *CreateSQLCommand*, onde através de expressões regulares, são feitas junções de outras tabelas com os campos de informações considerados mais relevantes. Desta junção surge uma instrução SQL mais complexa, executada através do processador *ExecuteSQL*, e o resultado é encaminhado para o *SplitAvro*, o qual divide os dados em registos (tuplos de informação), que depois são transferidos de forma sequencial para o *ConvertAvroToJson*, que tal como o próprio nome indica, converte os dados do formato *Avro* para Json. Feito isto, os dados são finalmente publicados no *Kafka* através do processador *PublishKafka*. E para monitorizar os dados, foi utilizado o *LogAttribute* para verificar se os atributos dos

dados foram transferidos com sucesso, ou se por algum motivo ocorreu alguma inconsistência nos dados.

5.5. Configuração da Capacidade do Serviço de Indexação

A medida que os dados são publicados no *broker*, estes são lidos pelo serviço de indexação *kafka*, sendo colocados automaticamente disponíveis para consulta através do *Druid*. Este serviço de indexação apesar de ser executado no *Druid*, utiliza o próprio mecanismo de partição e *offset* do *Kafka* para detetar e ler novos eventos de dados, recorrendo a tarefas de indexação criadas no *Druid* e executadas de forma continua. As tarefas de indexação *kafka* estavam limitadas aos recursos disponíveis no único nó de *middle manager* utilizado. Por isso foi preciso garantir que o mesmo tivesse capacidade suficiente, configurando a propriedade *druid.worker.capacity* para suportar as opções de configuração definidas para os supervisores. Atendendo que o *Druid* só permite utilizar um supervisor por cada fonte de dados, e neste caso como foram definidas três (auto, despacho e requerimento) no *kafka*, consequentemente foi configurada o mesmo número de supervisores para o serviço de indexação em tempo real. Com efeito, a capacidade de trabalho deve ser definida no mínimo com o valor de três ou múltiplos do mesmo, permitindo assim que estejam pelo menos três tarefas de indexação em tempo real, sendo executadas simultaneamente um para cada supervisor. Caso esta propriedade seja mal dimensionada, as tarefas de indexação do *kafka* serão enfileiradas e aguardarão até que uma das tarefas em execução termine.

Uma tarefa de indexação irá permanecer no modo de leitura enquanto o tempo definido em *taskDuration* (duração de tarefa) não expirar, momento pelo qual transita para o estado de publicação dos dados da memória para o *Deep Storage*. Ao passo que uma tarefa irá permanecer no estado de publicação, o tempo necessário para criar os segmentos, transferi-los para o *Deep Storage*, e tê-los carregados e prontos no nó de dados históricos. Entretanto, a tarefa de indexação é abruptamente terminada, caso termine o período de tempo (*completionTimeout*) definido para aguardar antes de declarar uma falha na tarefa de publicação. Em geral, a quantidade de tarefas de leitura é dada pela multiplicação entre os valores definidos em réplicas e o número de tarefas (*taskcount*). A exceção é quando o número de tarefas é maior do que o número de partições do respetivo tópico criado no

kafka, neste caso será utilizado o número de partições. Por essa razão, os tópicos criados no *kafka* foram estendidos para duas partições distribuídas por dois *brokers*, sendo que este valor pode ser reconfigurado dependendo das necessidades. O que implica que para este cenário de implementação, o número de tarefas no máximo só pode ser estendido até dois por cada supervisor.

Na Figura 21, são apresentadas algumas das especificações utilizados para configurar um dos supervisores.

```
    "granularitySpec": {
      "type": "uniform",
      "segmentGranularity": "day",
      "queryGranularity": "hour"
    }
  },
  "tuningConfig": {
    "type": "kafka",
    "maxRowsPerSegment": 5000000,
    "intermediatePersistPeriod": "PT5M",
    "handoffConditionTimeout": 60000,
    "basePersistDirectory": "/tmp",
    "resetOffsetAutomatically": true,
    "workerThreads": 4,
    "chatThreads": 4,
    "shutdownTimeout": "PT80S"
  },
  "ioConfig": {
    "topic": "autopendente",
    "consumerProperties": {
      "bootstrap.servers":"192.168.160.65:9092, 192.168.160.65:9093",
      "auto.offset.reset": "earliest"
    },
    "taskCount": 1,
    "replicas": 1,
    "taskDuration": "PT25H",
    "completionTimeout": "PT1M",
    "useEarliestOffset": true,
    "skipOffsetGaps": true
  }
```

Figura 21 - Configuração do supervisor de indexação para o Auto

Como se pode constatar na especificação de configuração do serviço de indexação, o valor definido por hora para o caso da granularidade das *queries* (*queryGranularity*), é a mesma utilizada na especificação das *queries* efetuadas no *Druid*, isto evita processamento extra no momento de retornar os resultados, sendo que os dados já são previamente agrupados segundo a granularidade pela qual os resultados das *queries* são retornadas. Apesar de um valor mais alto para a granularidade tornar este processo mais rápido, dado que mais dados são carregados para a memória, contudo, esse valor esta limitado pela quantidade de

memória disponível. Ao passo que, a granularidade dos segmentos (*segmentGranularity*) está diretamente relacionada com o número de vezes que é efetuada a escrita dos dados para o disco, e consequentemente com o tamanho dos segmentos. Quanto maior a granularidade dos segmentos, maior será o tamanho dos ficheiros produzidos, portanto, menos vezes é realizada a escrita para o disco, o que faz com seja menor o número de fragmentos de ficheiros armazenados.

A granularidade dos segmentos para cada supervisor foi configurada com uma hora a menos em relação ao valor especificado na duração das tarefas de indexação (*taskDuration*). Isto permite que, após o término do prazo definido na granularidade dos segmentos, o serviço de indexação ainda tenha tempo de executar tarefas adicionais e necessárias para a publicação dos segmentos no *Deep Storage*. Esse tempo adicional é dado pela diferença entre o valor definido na duração das tarefas e a granularidade dos segmentos.

As opções de configurações definidas contribuem para o aumento do desempenho das *queries* no *Druid*, e juntamente com outras especificações para os serviços de indexação, foram submetidas no *Overlord* por meio de um POST HTTP. Para cada um dos serviços de indexação configurados, ao serem submetidos no *Overlord* é automaticamente criado um supervisor que monitoriza em tempo real e de forma continua a entrada de eventos de dados provenientes do *kafka*. Recorrendo a consola de coordenação dos serviços de indexação, acedido normalmente através do endereço IP do *Overlord* na porta 8090, foi possível visualizar e fazer a gestão dos supervisores e tarefas em execução (Figura 22).

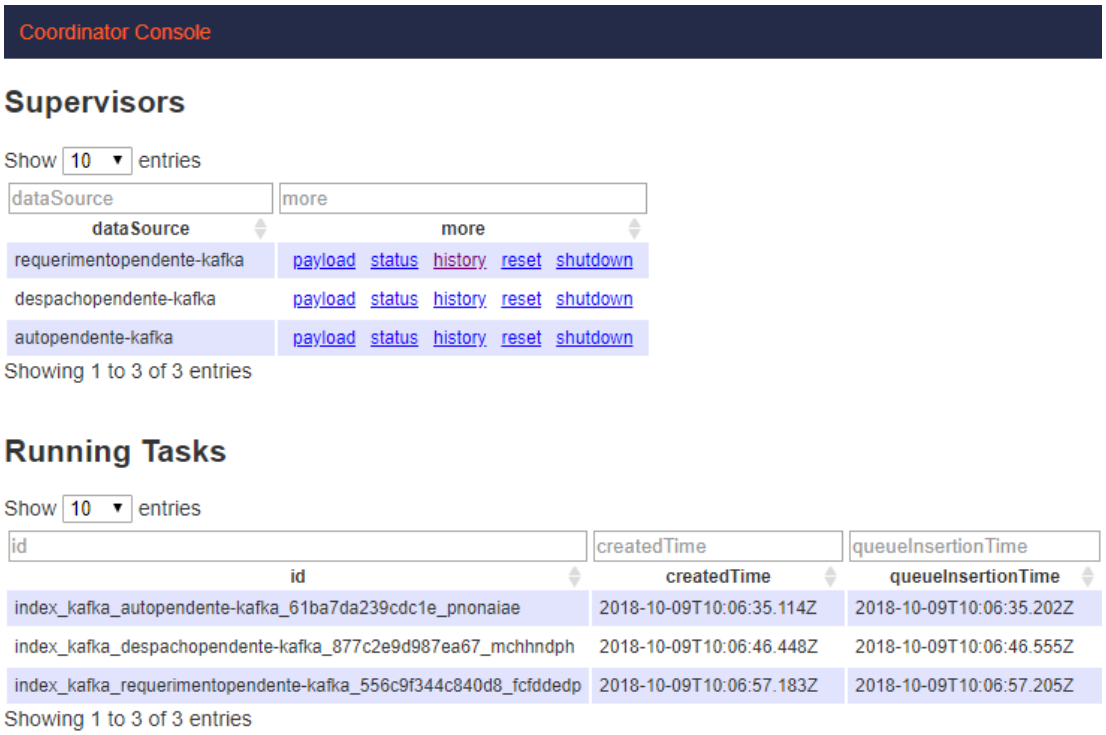

Figura 22 - Consola web de coordenação do serviço de indexação

Terminado a duração de uma tarefa de indexação, tendo sido configurado para vinte e cinco horas, os dados indexados em memória relativamente a esta tarefa com um prazo de um dia são transformados num segmento, o qual é posteriormente transferido para o *Deep Storage*. Para este caso em particular, o *Deep Storage* foi configurado para armazenamento local, que é o padrão do *Druid*, porque na fase de implementação entendeu-se que este tipo de armazenamento atende aos requisitos do sistema em causa. Parcialmente, esta decisão de utilizar armazenamento local está relacionada com a limitação de recursos disponíveis para colocar uma infraestrutura deste tipo em funcionamento, sendo que um dos objetivos é reduzir os custos ao mínimo possível sem comprometer o desempenho do sistema.

Através da consola de gestão do *Druid*, é possível visualizar um exemplo da linha do tempo (*timeline*) dos segmentos armazenados no *Deep Storage* com tamanhos de um dia respetivamente. Tal como apresentado na Figura 23 são vários os segmentos armazenados, neste caso em concreto, com dados desde 6 de fevereiro de 2014 até 16 de dezembro de 2017, os quais perfazem um total de 4,58 megabytes.

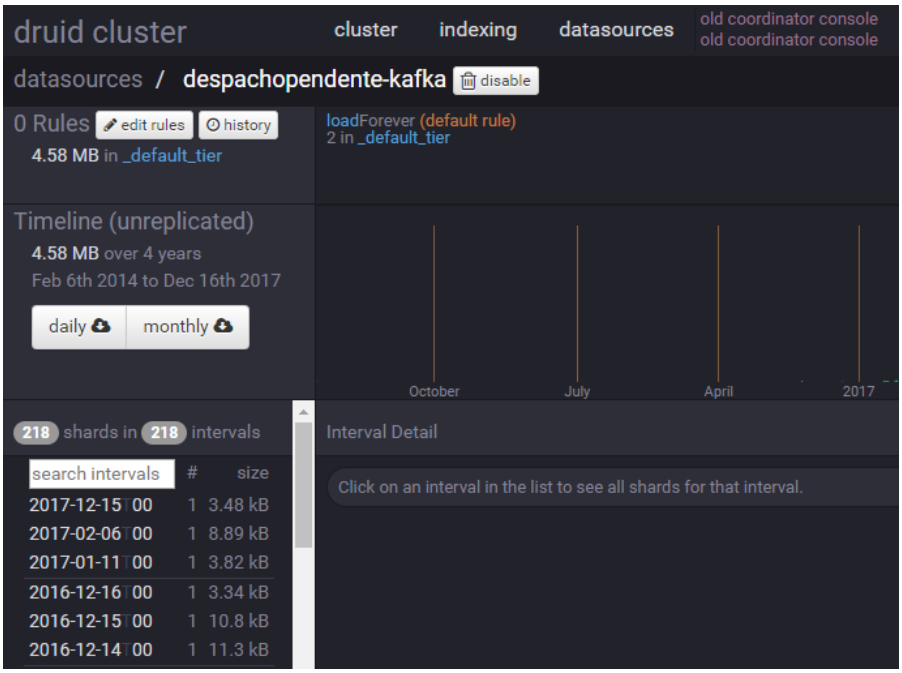

Figura 23 - Consola de gestão do Druid

Também existem outras opções de *Deep Storage* suportadas pelo *Druid* e muito aconselhadas para sistemas em produção, sendo as mais utilizadas o *Hadoop* e o armazenamento S3 na *cloud*, os quais podem ser facilmente configurados com o mínimo de alterações nos componentes do *Druid*. Sendo que, muitas das opções de configuração tomadas neste processo de configuração, estão intrinsecamente ligadas aos recursos computacionais disponíveis.

5.6. Implementação da interface entre o SIJ e o Data Warehouse

As *queries* submetidas diretamente no *Druid* são difíceis de implementar, dado que utilizam uma estrutura própria de dados, fora do padrão SQL normalmente utilizado. Por isso foi construída uma *interface*, que permite converter as *queries* nativas do *Druid*, como o *Select*, *TimeSeries*, *TopN* e *GroupBy*, numa estrutura de classes de objetos mais facilmente manipuláveis através do SIJ.

Na Figura 24, é apresentada um exemplo da estrutura de dados de uma *query* nativa do *Druid*, do tipo *TimeSeries*, onde é aplicado um filtro a um campo de dados e a agregação do tipo *longSum*. Apesar de ter sido utilizado a contagem (*longSum*) na especificação da agregação, é também possível utilizar outros pré-cálculos como média, mínimo, máximo

entre outros. Além disso, existe a possibilidade de adicionar outros elementos que fazem parte da especificação deste tipo de *queries*, tal como em outros exemplos de *queries* utilizados, onde foi acrescentado mais um elemento designado de pós agregação. Na especificação da pós-agregação podem-se efetuar várias operações, como por exemplo cálculos aritméticos (soma, diferença, multiplicação e divisão), utilizando os valores pré-calculados na agregação.

```
{
  "queryType": "timeseries",
  "dataSource": "autopendente-kafka",
  "intervals": [
    "2013-01-01/2018-07-01"
  ],
  "granularity": "hour",
  "filter": {
        "type": "selector",
        "dimension": "DescritorDoAuto",
        "value": "421A79D4-AF1E-47C9-9C60-444BF104D98D"
        },
  "aggregations": [
    {
      "type": "longSum",
      "fieldName": "countauto",
      "name": "auto_with_descritor"
    }
   ]
}
```

Figura 24 - Estrutura de uma query do tipo TimeSeries

O principal objetivo da construção desta *interface* é de basicamente permitir a aplicação *web* do SIJ interagir com os dados armazenados no *Data Warehouse*. Esta *interface* consiste num conjunto de funções, construídos e adaptados ao SIJ, podendo ser facilmente convertidas em serviços. As funções encontram-se agrupadas em blocos de acordo com as funcionalidades de cada uma delas. O primeiro bloco é responsável por traduzir estruturas de *queries* típicas do *Druid*, tal como apresentado na Figura 24, para uma estrutura de classes de objetos. No segundo bloco, estas classes de objetos são utilizadas para construir a *query* pretendida, por intermédio de funções que recebem como parâmetros os dados necessários para subscrever os valores dos atributos desses objetos. Na Figura 25, é apresentado uma parcela de uma função onde é feita a conversão de um filtro de uma *query* nativa do *Druid*, para uma estrutura de objetos.

```
tsm.aggregations = new List<Aggregator>()
{
    new FilteredAggregator()
    {
        filter = new SelectorFilter()
        {
            dimension = aggregator_dimension,
            value = null
        },
        aggregator = new CountAggregator()
        {
            name = "count_NullAggregator"
        }
    },
```

Figura 25 - Transformação de queries numa estrutura de objetos.

Com a *query* estruturada e preenchida com todos os parâmetros necessários, esta é enviada para o próximo bloco, o qual é responsável por convertê-la para o formato Json, e submetê-la para o *Druid* através do endereço IP e da porta pela qual o *Overlord* foi configurado (Figura 26).

```
public static async Task<string> PostQuery(string url, QueryModel query)
{
    HttpClient client = new HttpClient();

    client.BaseAddress = new Uri(url);
    client.DefaultRequestHeaders.Accept.Clear();
    client.DefaultRequestHeaders.Accept.Add(new MediaTypeWithQualityHeaderValue("application/json"));

    JsonSerializerSettings settings = new JsonSerializerSettings
    {
        NullValueHandling = NullValueHandling.Ignore
    };

    string json = JsonConvert.SerializeObject(query, Formatting.None, settings);
    StringContent content = new StringContent(
        json,
        Encoding.UTF8, "application/json");
    HttpResponseMessage response = client.PostAsync("druid/v2/?pretty", content).Result;
    string result = await response.Content.ReadAsStringAsync();

    if (response.IsSuccessStatusCode)
    {
        System.Diagnostics.Debug.WriteLine("Data posted");
        return result;
    }
```

Figura 26 - Conversão e envio da query para o Druid

Como resultado da execução da *query* é obtido um Json, sendo o mesmo convertido novamente para uma estrutura de classes de objetos específica para cada tipo de *query*, através do qual pode-se facilmente aceder aos atributos desse objeto que representam a estrutura da *query* em questão.

Para confirmar se a *interface* desenvolvida realmente funcionava conforme os requisitos estabelecidos, optou-se por construir uma aplicação *web* MVC, para efetuar pesquisas sobre os dados do SIJ armazenados no *Druid*. Apesar de ser uma aplicação com poucos recursos permite efetuar todos os tipos de consultas suportadas pela *interface* desenvolvida. Além de que, esta aplicação foi desenvolvida utilizando a mesma linguagem de programação pelo qual o SIJ foi desenvolvido, e utilizando o mesmo conceito de acesso aos dados, com o intuito de validar a usabilidade da *interface* para que depois o mesmo possa ser facilmente integrado no SIJ.

Na Figura 27, é apresentada uma tabela extraída da aplicação *web* desenvolvida, e resultante da execução de uma *query* do tipo *TopN*. Os resultados apresentados na tabela são agregações ou pré-cálculos do tipo *count* (contagens), feitas para determinar o número total de tarefas pendentes relativas aos autos, despachos ou requerimentos. Neste caso, o total de tarefas pendentes foi calculado para cada utilizador do SIJ que submeteu um ou mais despachos. Com esse tipo de informação, estando a tabela ordenada por ordem decrescente em relação ao número de tarefas, consegue-se facilmente determinar qual é o utilizador com mais tarefas pendentes.

Contagem Sem Explicador	Contagem Com Explicador	Quem Submeteu	Total de Pendentes
7832	4083	980E360D-A67A-4B8E-9CA9-5509C39AC6DB	3749
4673	2579	1B3E7767-9C67-4FC1-B0B4-A0C659696D75	2094
2127	1562	CB915905-97C1-4303-9D41-A927FD6E6D1C	565
2621	1129	55753941-1123-48C5-8682-7C6FF77DFEC0	1492
1131	911	D6D5A2AD-915A-490B-9620-A37558DCAC47	220
389	290	D3DD7EF0-81E6-42E9-9D04-69E1C32FF74F	99
318	289	792086F0-0689-4881-B460-CBC4AC070C23	29
315	261	1CF7851E-78B1-4704-81AB-DFE50B7D9DFB	54
85	51	31142742-ABD8-4DD2-BD97-0C48EFCEDA1D	34
31	26	E65720E4-4A27-4883-AE55-EB1459CE1FDA	5
9	9	A527F778-5B15-48E4-B079-FC1555A15FE2	0
23	9	F24C9707-3D52-4563-993F-66FB7243C63F	14
31	2	1D2D575B-663D-4E25-97B9-5C7B04C913B9	29

Figura 27 - Resultados da execução de uma query do tipo TopN

É de realçar que a *interface* desenvolvida foi também utilizada com um conjunto de dados designada de *PageViews*, normalmente utilizada para testes, tendo funcionando normalmente, tal como funcionou para o SIJ. Portanto, pode-se concluir que esta *interface*,

devido sobretudo a sua modularidade, pode ser utilizada com sucesso para outros conjuntos de dados, desde que estes mesmos dados sejam previamente armazenados no *Druid*.

5.7. Configuração do Metabase como ferramenta de apoio a decisão

O *Metabase* é uma ferramenta de visualização de dados simples de configurar, com uma *interface* gráfica fácil de utilizar e bastante apelativa. Esta ferramenta oferece três formas de implementar *queries* e obter dados das bases de dados pelo qual esta conectado: através de métricas previamente criadas, do construtor de *queries* personalizadas, ou do editor de *queries* nativas da base de dados utilizada. Neste caso o editor de *queries* nativas foi o método utilizado para obter os dados do *Druid*, sendo os resultados posteriormente apresentados na forma de tabelas ou gráficos, dependendo da opção selecionada. Pois, o *Metabase* oferece a possibilidade de selecionar um dos diversos tipos de visualização de dados suportados por esta ferramenta. Visto que as *queries* nativas do *Druid* são construídas utilizando a estrutura de um Json, também no editor de *queries* do *Metabase* teve-se que utilizar a mesma estrutura de dados. Na Figura 28, é apresentado um exemplo de uma *query* na estrutura de um Json, tendo o mesmo sido executado no *Metabase* graças ao editor de *queries* nativas.

```
{
  "intervals": "2013-08-15T00:00:00+00:00/2018-08-15T16:23:43+00:00",
  "threshold": 50000,
  "dataSource": "autopendente-kafka",
  "postAggregations": [],
  "dimension": "NumSeqCalculado",
  "granularity": "all",
  "aggregations": [
    {
      "type": "count",
      "name": "countauto"
    }
  ],
  "metric": "countauto",
  "queryType": "topN"
}
```

Figura 28 - Exemplo de uma query nativa do Druid

Ainda, com recurso ao *Metabase* foram construídas e armazenadas muitas outras *queries*, as quais, depois foram utilizadas para criar um *dashboard* com dados do SIJ provenientes do *Druid*, utilizando os mais variados tipos de visualização suportadas pelo *Metabase*, tal como apresentado na Figura 29.

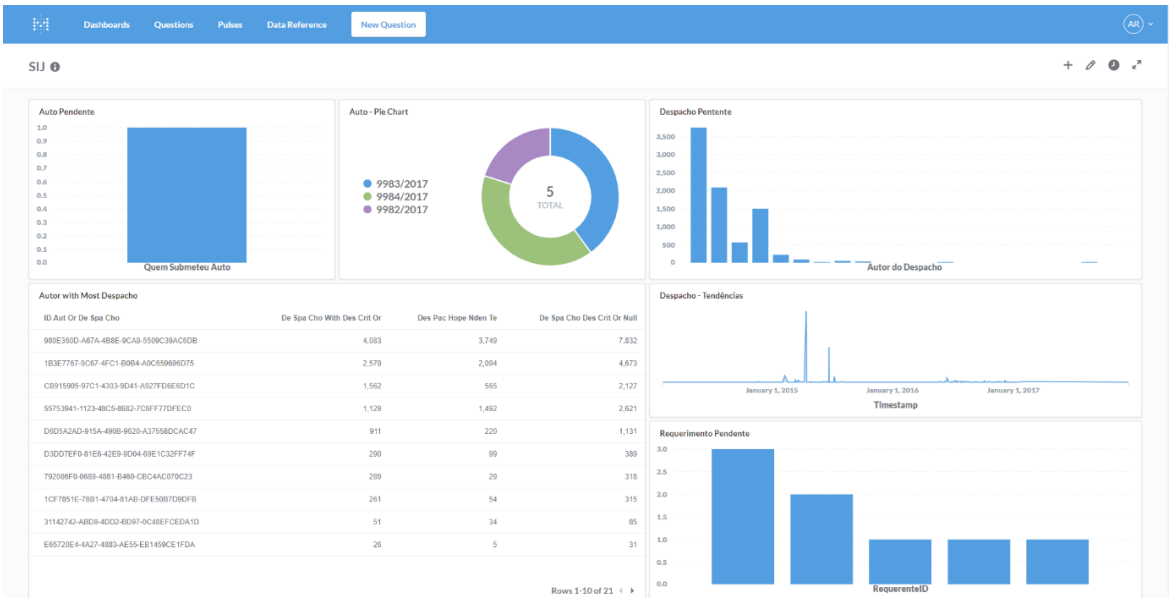

Figura 29 - Dashboard construído no Metabase com dados do SIJ

As visualizações na *dashboard* foram construídas principalmente para que se consiga de forma rápida e com o mínimo de esforço determinar o total de Autos, Despachos e Requerimentos para cada utilizador que os submeteram através do SIJ, facilitando as tomadas decisões de forma informada. Caso nenhuma dessas *queries* vá ao encontro das pretensões do decisor, este pode sempre adicionar outras *queries* de forma simples e eficiente, acoplando essas ou outros tipos de visualizações suportadas pelo *Metabase*.

6. Testes e Resultados

Para comprovar o desempenho das soluções adotadas na implementação da arquitetura deste sistema de *Data Warehouse* em tempo real, foi realizado um conjunto de testes com especial preponderância nos testes de latência para diferentes cargas de dados. Entretanto, dada a dificuldade para encontrar ferramentas de *benchmarking*, que possam ser aplicados ao nível de toda a arquitetura do sistema como um único bloco, optou-se por efetuar testes a cada um dos componentes da arquitetura separadamente e no fim, se possível, juntar todos os resultados obtidos num único valor que represente o desempenho de todo o sistema.

Visto que cada componente da arquitetura se encontra instalado e configurado numa máquina com características diferentes, dependendo das necessidades de utilização, assim também os testes estarão limitados aos recursos configurados nestas respetivas máquinas:

- ✓ O CDC está sendo executado na máquina virtual onde se encontra instalada a base de dados do SIJ, cuja especificação é de 2 VCPU, 8 GB de RAM, 225 GB de armazenamento no disco, e 10 Gbps de Ethernet;
- ✓ O *Apache Nifi* e o *Apache Kafka* foram instaladas em máquina virtuais com 1 VCPU, 4 GB de RAM, 100 GB de armazenamento no disco, e 100Mbps de Ethernet respetivamente;
- ✓ Todos os cinco nós do *Druid* (*coordinator*, *historical*, *overlord*, *middle manager*, *broker*) foram configurados numa única máquina virtual com 1 VCPU, 64 GB de RAM, 100 GB de armazenamento no disco, e 100Mbps de Ethernet;

- ✓ Por último, o API foi implementado e executado numa máquina física com um processador Core i7-6700, 8 GB de RAM, 250 GB de armazenamento no disco, e 1 Gbs de Ethernet.

Esta forma de realizar os testes permite obter o desempenho minucioso de cada um dos componentes da arquitetura, o que também traz benefícios para quem pretende incorporar algum destes componentes na arquitetura de um sistema semelhante.

6.1. Obtenção do Tempo consumido na Captura dos Dados

O desempenho da captura de dados é normalmente dado pela diferença entre o momento em que uma transação de manipulação de dados ocorre numa das tabelas monitoradas, e o instante em que essa mesma transação é refletida nas tabelas de alteração do CDC. Esse tempo que os dados demoram a serem refletidas nas tabelas CDC é designado de latência. Quando o CDC consegue acompanhar a carga de trabalho, a latência normalmente será menor do que o *pollinginterval* configurado nas tarefas de captura de dados. A latência foi determinada utilizando a vista de gestão dinâmica *sys.dm_cdc_log_scan_sessions*. Esta vista de gestão dinâmica, além da latência permite determinar qual foi o tempo gasto pelas sessões de scan, e quantos comandos e transações foram processados. A coluna de latência é preenchida apenas para sessões ativas. Para sessões com um valor maior que zero na coluna empty_scan_count, a coluna de latência é atribuída o valor zero. A *query* apresentada a seguir retorna à latência média das sessões mais recentes:

```
SELECT latency FROM sys.dm_cdc_log_scan_sessions WHERE session_id = 0
```

Os dados de latência são utilizados para determinar o quão rápido ou lento o processo de captura de dados está processando as transações. Estes dados são mais úteis quando o processo de captura está sendo executado continuamente, caso contrário, a latência pode ser alta devido ao atraso entre as transações que estão sendo confirmadas nas tabelas de origem e o processo de captura em execução. Na Figura 30, encontra-se representado um gráfico com os dados de latência em ordem ao número de registos inseridos nas tabelas de origem do auto, requerimento, despacho e processo.

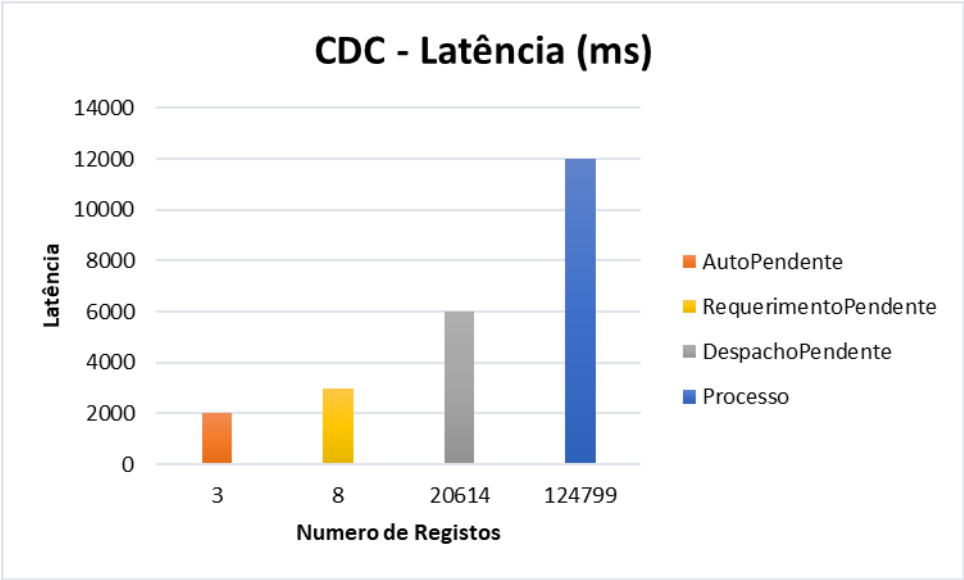

Figura 30 - Latência na captura dos dados

Outra medida importante da eficiência do processo de captura é o *throughput*. Esta medida traduz-se no número de comandos por segundo que são processados durante cada sessão. Portanto para determinar o *throughput* de uma sessão, foi feito a divisão entre o número de comandos executados e a respetiva duração. Valores esses que foram extraídos da vista de gestão dinâmica sys.dm_cdc_log_scan_sessions. A *query* apresentada a seguir retorna o *throughput* médio das sessões mais recentes:

```
SELECT command_count/duration AS [Throughput] FROM
sys.dm_cdc_log_scan_sessions WHERE session_id = 0
```

Na Figura 31, encontra-se representado na forma de gráfico o *throughput* de acordo com os registos inseridos nas tabelas de "DespachoPendente" e "Processo" respetivamente. Para este exemplo em específico só foram utilizadas estas duas tabelas, contrariamente ao gráfico de latência, porque apenas estas tabelas contêm registos em número suficientes para que se consiga fazer inserções de forma contínua com o objetivo de determinar os respetivos valores de *throughput*.

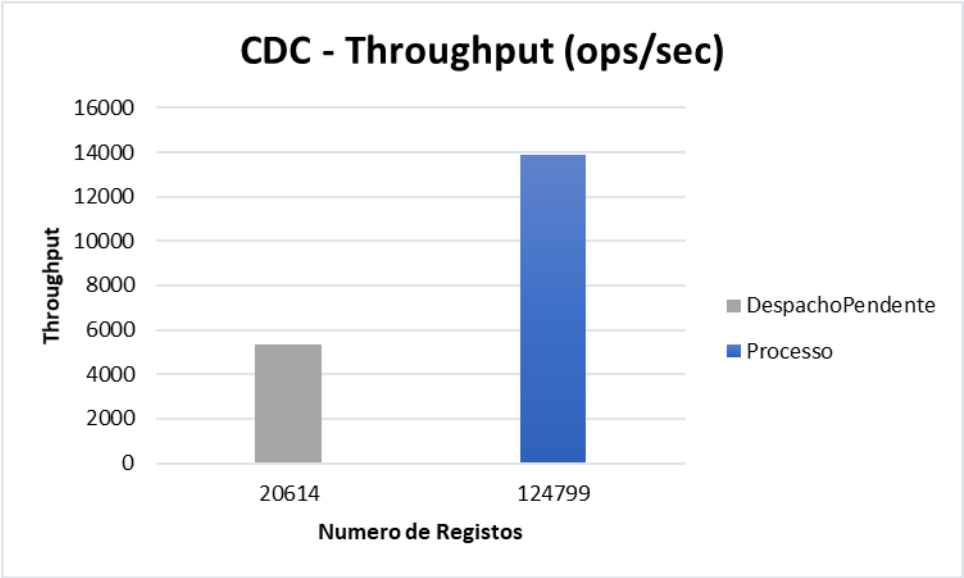

Figura 31 - Numero de operações por segundos na captura dos dados

Como se pode constatar em ambos os gráficos apresentados neste tópico, tanto os valores de latência como de *throughput*, são maiores para as tabelas que contém maior número de registos. Contudo isto afeta muito pouco o desempenho do CDC, o que se considera um bom resultado. Dado que apenas para valores muito grandes de registos inseridos de forma continua, é que estes valores começam a traduzir-se num ligeiro aumento dessas medidas.

6.2. Cálculo do Tempo consumido no processo ETL do Apache Nifi

Para monitorizar o fluxo de dados no *Apache Nifi*, é preciso determinar de alguma forma o que aconteceu com os dados em determinado ponto ou ao longo do fluxo implementado. *Data Provenance* (proveniência dos dados) fornece esta informação. A medida que os dados são extraídos, transformados, roteados, divididos, e distribuídos para os outros processadores, essa informação é armazenada e indexada no repositório de proveniência de dados do *Apache Nifi*, podendo ser pesquisados, e avaliados ao nível de otimização do fluxo de dados em tempo real. Por padrão o *Apache Nifi* atualiza essas informações a cada cinco minutos, porém isso pode ser alterado mediante algumas configurações.

Tendo a proveniência dos dados em qualquer etapa do fluxo de dados, sendo que os tempos de execução referentes a cada processador também são registados, torna-se mais fácil determinar a latência com que os dados fluem ao longo dos processadores. Assim, de

forma a conseguir transformar, distribuir e armazenar os dados da proveniência em uma base de dados ou ficheiros para posteriormente serem analisados, foi construído um fluxo de monitoramento paralelamente ao fluxo principal (Figura 32). O fluxo principal engloba três grandes grupos de processos: - autos, despachos e requerimentos, sendo cada um destes compostos por um conjunto de processadores.

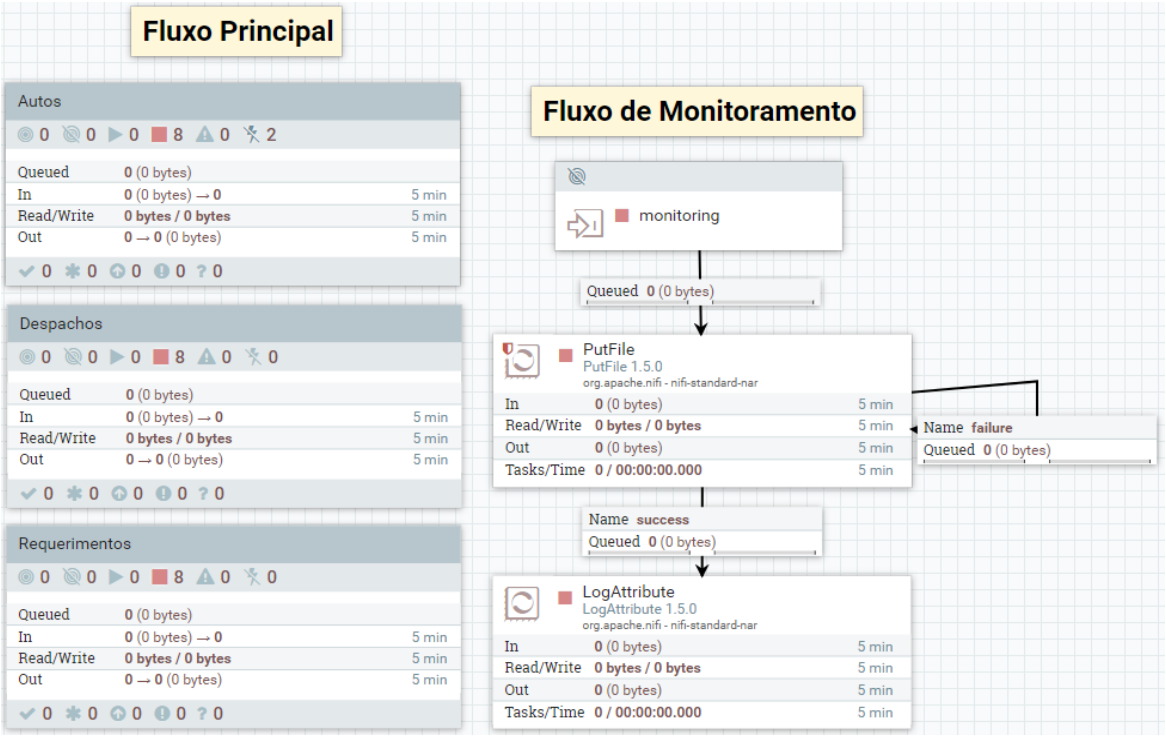

Figura 32 - Fluxo principal e fluxo de monitoramento

Antes dos dados de proveniência serem processados pelo fluxo de monitoramento, primeiramente foi utilizado uma tarefa de relatório *Site-to-Site*, denominado de *SiteToSiteProvenanceReportingTask*, para extrair esses dados e enviá-los para a mesma instância de *Apache Nifi*. Evidentemente, num ambiente de produção não é aconselhável criar um fluxo de monitoramento na mesma máquina onde está implementado o fluxo principal, visto que a tarefa de relatório *Site-to-Site* também permite enviar dados para instâncias de *Apache Nifi* diferentes. Porém, esta decisão de colocar os dois fluxos na mesma máquina foi no sentido de fazer uma melhor gestão dos recursos disponíveis, que são limitados.

Depois de extrair os dados de proveniência, os mesmos foram enviados para uma porta de entrada configurada no fluxo de monitoramento. Estes por sua vez foram processados e escritos em ficheiros num diretório previamente selecionado para depois serem analisados. Após uma análise detalhada dos dados de proveniência, foram extraídos os tempos de execução dos campos de informação *timestampMillis* e *lineageStart*, referentes a cada grupo de processos (auto, despacho e requerimento). Assim, para obter os valores de latência, foi feita a diferença entre o *timestampMillis* e o *lineageStart* para cada um dos grupos de processos. Com os resultados da latência para os autos, despachos, e requerimentos, foi construído um gráfico, onde se encontram ilustradas as variações da latência em função do número de registos inseridos nas respetivas tabelas criadas pelo CDC (Figura 33).

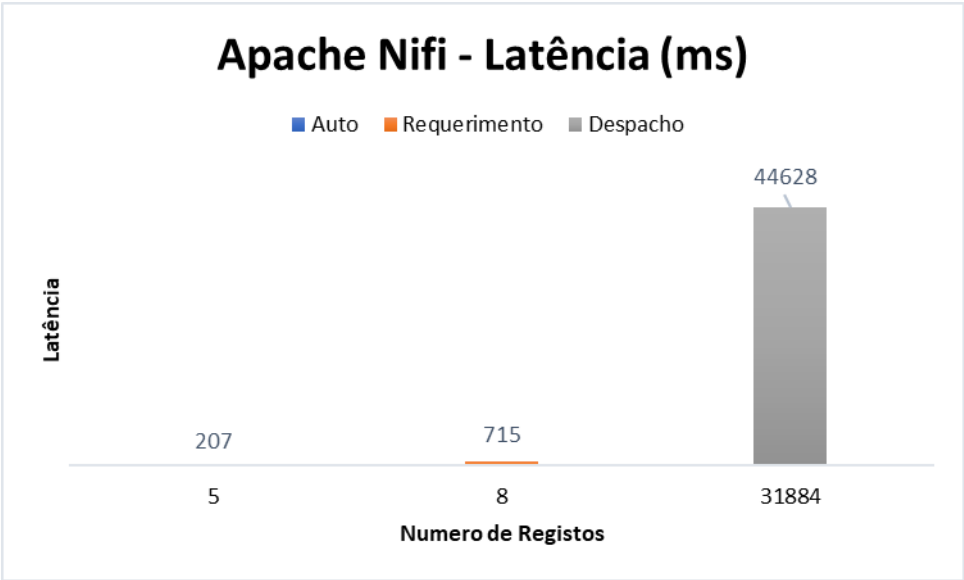

Figura 33 - Variação da latência em função do número de registos

Ao serem analisados os dados, tendo em consideração que a unidade de medida utilizada na determinação da latência é milissegundos, verifica-se que o valor máximo da latência é de aproximadamente 45 segundos para o processamento de 31884 registos, ficando ainda relativamente distante de atingir 1 minuto. Por isso, se considera-se os resultados da latência obtidos satisfatórios.

6.3. Testes de consumo de mensagens no Apache Kafka

Para que se consiga determinar o desempenho do consumidor *kafka* integrado no serviço de indexação do *Druid*, foi utilizado um script denominado "*kafka-consumer-perf-test.sh*", o qual por padrão já se encontrava incluído no diretório de instalação do *Apache Kafka*. Este script é executado através do terminal e recebe como parâmetros de entrada, o número de mensagens que serão consumidas, o nome do tópico pelo qual as mensagens foram publicadas, uma lista de *brokers* previamente configurados no *kafka*, entre vários outros parâmetros.

Como resultado da execução deste script, obteve-se um conjunto de dados relativos ao desempenho do *kafka* para diferentes quantidades de mensagens consumidas (Tabela 6).

Tópicos	***Auto***	***Requerimento***	***Despacho***
Numero de Mensagens	5	8	31884
Latência (ms)	279	342	1606
***Throughput* (Msg/sec)**	17,9211	23,3918	19196,7621
MB/sec	0,007	0,0108	7,7648

Tabela 6 - Desempenho no consumo de mensagens do Kafka

Utilizando os dados da latência em relação ao número de mensagens lidas para cada um dos três tópicos, foi construído um gráfico de barras através do qual consegue-se tirar melhores ilações sobre o comportamento do *kafka* (Figura 34).

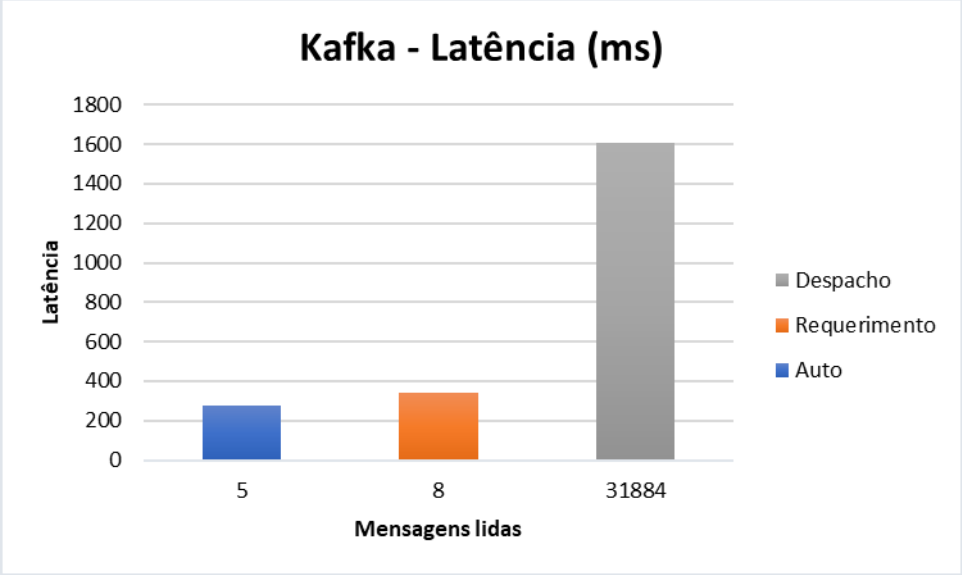

Figura 34 - Variação de Latência em relação ao número de mensagens

A latência no que concerne aos despachos é muito superior comparando com valores obtidos para os autos e requerimentos, isto porque a quantidade de mensagens lidas também grande. Tendo em consideração a quantidade de mensagens lidas, e os valores altos apresentados para o despacho em relação a quantidade de megabytes processados e o *thoughput* utilizado, pode-se afirmar que o valor de latência obtido é aceitável. Isto torna-se ainda mais evidente através da normalização desses valores, ou seja, dividindo a latência obtida pelo número de registos para cada um dos três tópicos (auto, despacho e requerimento). Como resultado desta normalização é apresentado um gráfico na Figura 35, onde se encontra ilustrado a tendência da latência a medida que quantidade de registos for aumentando.

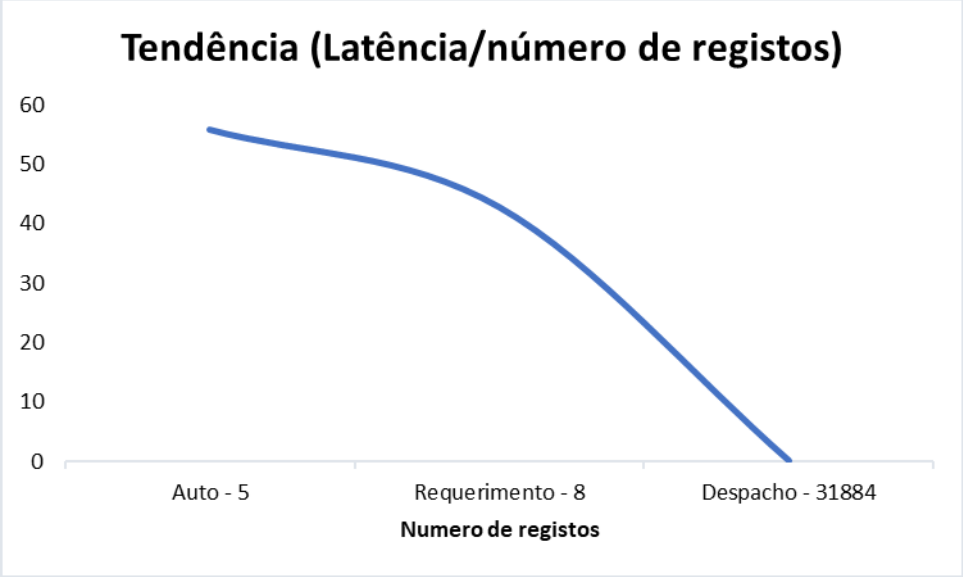

Figura 35 - Tendência da latência quando o número de registos aumenta

De acordo com o gráfico acima apresentado, pode-se constatar que a razão entre a latência e o número de registos tende para zero a medida que a quantidade de mensagens nos tópicos forem aumentando. Isto significa que a latência cresce de forma muito mais lenta em relação a quantidade de mensagens consumidas, comprovando o bom desempenho do Kafka independentemente da quantidade de mensagens lidas.

6.4. Benchmarking utilizando o YCSB-TS

É difícil encontrar ferramentas de *Benchmarking* adequados para efetuar testes de desempenho nos sistemas de bases de dados *NoSQL*, em parte porque os recursos de um e de outro diferem muito entre eles, e por outro lado porque não há uma maneira fácil de comparar o desempenho deste tipo de sistemas. Para solucionar, este problema investigadores da *Yahoo* desenvolveram um *framework* genérico e extensível, o qual designaram de *Yahoo Cloud Serving Benchmark* (YCSB), com o objetivo de facilitar comparações de desempenho da nova geração de sistemas de base dados (*NoSQL*), especialmente para o processamento de grandes volumes de dados.

Esta ferramenta é composta essencialmente por um cliente YCSB para cada um dos sistemas de bases de dados suportados, e as respetivas definições de cargas de trabalho principais (*workloads*). Qualquer um destes clientes possui funções para carregamento,

leituras, escritas e pesquisas de dados específicos para cada base de dados, os quais são responsáveis por executar as cargas de trabalho. Embora as principais cargas de trabalho forneçam uma imagem completa do desempenho de um sistema, o cliente é extensível para que se consiga definir novas cargas de trabalho com o intuito de examinar aspetos do sistema ou cenários da aplicação que não estão abrangidos pelas cargas de trabalho principais. Também podem ser criados novos clientes de modo que este *framework* possa suportar *benchmarking* de outras bases de dados chave-valor [75].

Por forma a preencher a lacuna que ainda existia quanto a ferramentas *open source* de *benchmarking* adaptadas a sistemas de base de dados de series temporais, foi desenvolvido uma extensão deste *framework* denominado de YCSB-TS, para medir o desempenho dessas bases de dados. Para conseguir isso, foram acrescentadas novas opções de cargas de trabalho e ligações para vários sistemas de bases de dados baseadas em series temporais.

6.4.1. Testes de Latência no Druid

Para que se consiga ter uma noção do desempenho do *Druid*, é essencial determinar a latência com que os dados são processados por este sistema de armazenamento. Com recurso ao YCSB-TS é possível determinar entre vários outros parâmetros, a latência de operações como a leitura, pesquisa e pré-cálculos são efetuadas no *Druid*, mediante a especificação de cargas de trabalho. Uma das cargas de trabalho utilizadas é o *workloada*, que engloba operações como o carregamento de dados (*Load*), operações de leituras (*read*) e remoção dos dados (*cleanup*). Neste caso, foram carregados para o *Druid* 31884 registos, os quais correspondem ao número total de despachos armazenados num dos tópicos criados no *Kafka*. Outra das cargas de trabalho utilizadas é o *workloadb*, o qual realiza pesquisas (*scan*) e operações habitualmente definidas como pré-cálculos ou agregações (*avg*, *sum*, *count*). Ao todo foram efetuados três testes no *Druid* sob as mesmas condições, um para carregamento dos dados e os outros dois para operações de leituras, agregações e pesquisas dos dados.

Obtidos os resultados da execução dos testes, foram construídos gráficos para facilitar a leitura dos dados, permitindo comparar a latência para diferentes tipos de *queries* suportados pelo *Druid*. Como se pode verificar pelo gráfico apresentado na Figura 36, a latência máxima atinge valores mais altos para operações de leitura (*read*), seguido de

cálculos da média (*avg*). No caso de operações de leitura do tipo *SELECT*, o valor alto obtido para a latência máxima é devido ao facto dos resultados serem paginados e ordenados segundo um ou mais campos de informações, sendo que isso exige um número considerável de comparações. Ao contrário do que acontece nas pesquisas do tipo *SCAN* que retornam os resultados sem uma estrutura definida (*RAW*) no modo *streaming*.

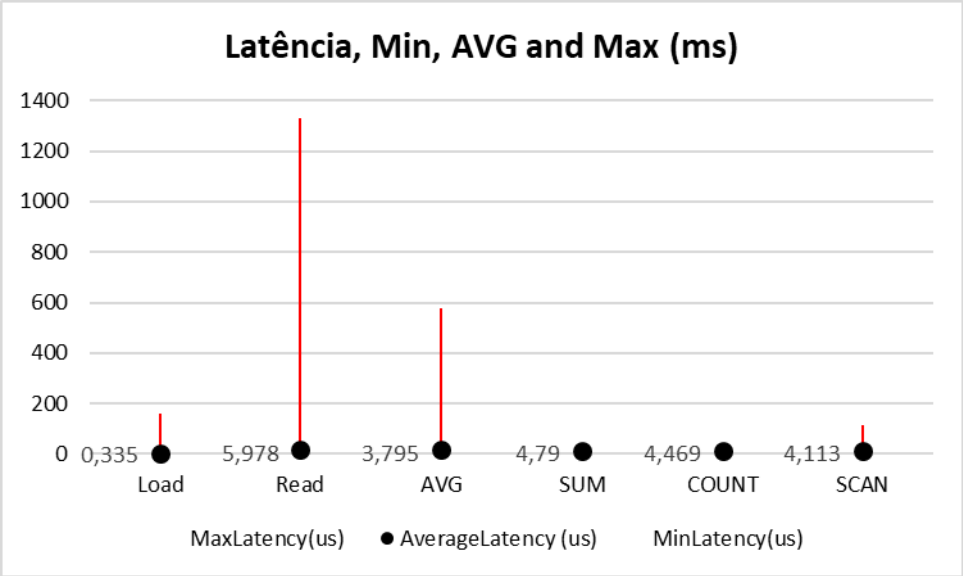

Figura 36 - Latência das principais operações executadas no Druid

Os valores máximos apontados no gráfico podem não corresponder totalmente ao que acontece, por isso também foi construído um gráfico comparando os valores 95 e 99 percentil da latência (Figura 37).

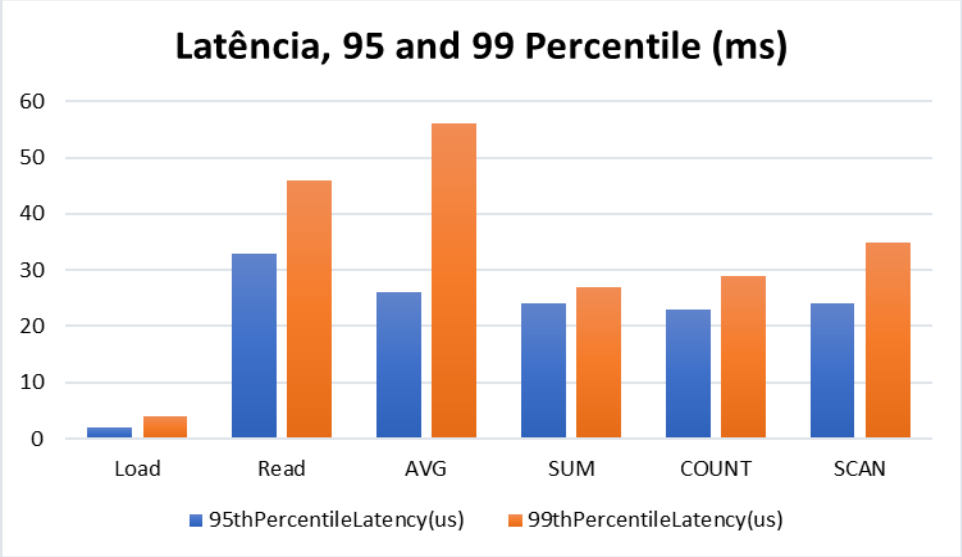

Figura 37 - Latência 95 e 99 percentile

Tendo em mente o gráfico acima representado, o que mais se destaca é o valor 99 percentil da latência para a média (*avg*), o qual corresponde aproximadamente a soma das latências de agregações do tipo somatórios (*sum*) e contagens (*count*), visto que normalmente para o cálculo da média é necessário primeiramente calcular esses dois valores.

6.5. Latência dos pedidos efetuados através da interface web

Utilizando as ferramentas de desenvolvedor do *chrome*, mais propriamente na área relativa a rede, foi medido o tempo com que os resultados das *queries* são retornados através da *interface web* desenvolvida (Figura 38). Este teste foi feito dentro de uma intranet, sem considerar os possíveis problemas sujeitos ao tráfego na internet, tendo sido desenvolvida uma aplicação *web* MVC especificamente para mostrar os resultados das *queries* efetuados através da interface *web*. Permitindo assim determinar de forma aproximada a latência com que as principais *queries* suportados pelo *Druid* foram efetuadas. Neste caso, o valor da latência apresentado é referente a uma *query* do tipo *TOPN* submetido sobre uma fonte de dados no *Druid* com 31884 registos sobre os despachos.

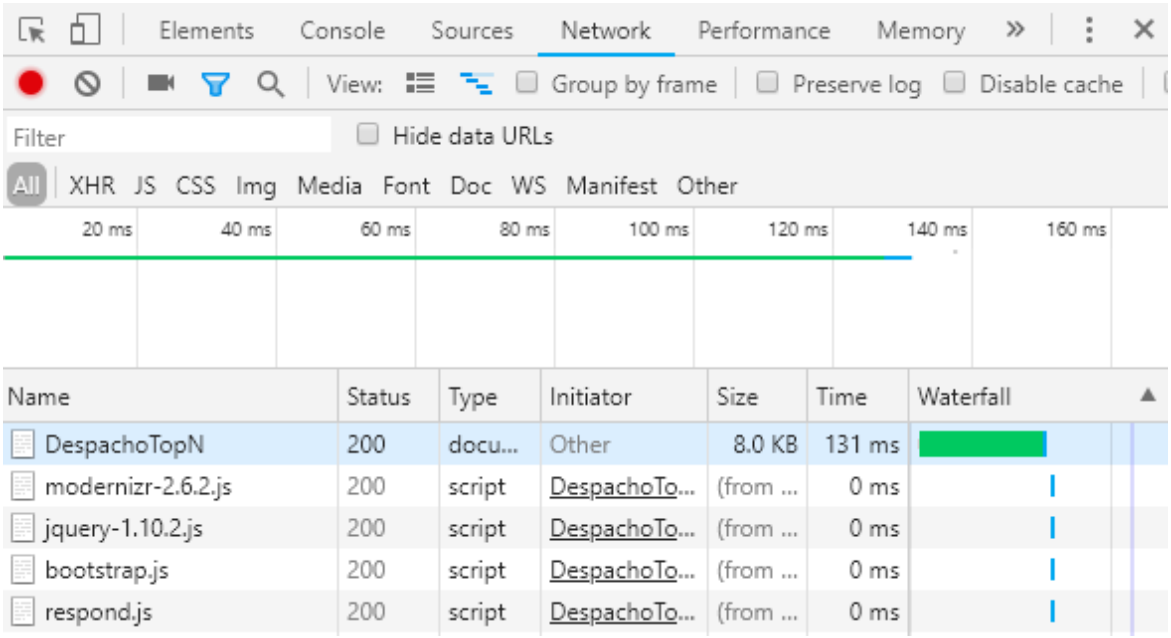

Figura 38 - Latência na consulta dos dados através da API

O mesmo tipo de teste foi aplicado a outros tipos de *queries* suportados pelo *Druid*, tendo obtido um valor de latência mais elevado para as *queries* do tipo *GROUPBY*. Dado que este tipo de *query*, além de efetuar agregações também agrupa os resultados segundo um ou mais dimensões. Portanto, se o objetivo é retornar as agregações (contagens, somatórios, médias) agrupados para uma única dimensão, então por questões de desempenho é melhor utilizar uma *query* do tipo *TOPN* em detrimento de um *GROUPBY*, pois este retorna os resultados de uma forma mais rápida.

6.6. Considerações finais

Tendo em vista os valores obtidos para latência, em relação a captura de dados alterados, ao processo ETL, ao consumo de dados provenientes do *kafka*, ao carregamento dos dados no *Druid*, e a consulta desses mesmos dados através da API desenvolvida, é possível agregar todos esses valores para cada fonte de dados num único valor que represente de forma aproximada, o desempenho de toda a infraestrutura implementada para os dados do SIJ. Por exemplo, somando os tempos de latência relativos aos despachos obtidos através dos testes efetuados a cada um dos componentes desta infraestrutura, conseguiu-se obter uma aproximação do tempo necessário para o processamento da totalidade dos despachos

inseridos, desde a captura dos dados até a apresentação dos resultados através da interface de programação de aplicação (API) (Tabela 7).

Latência para o processamento de 31884 despachos (ms)					
CDC	*Apache Nifi*	Consumidor *Kafka*	*Druid* (*Load*)	*TopN* (API)	**Total**
6000	44628	1606	30527,308	131	**82892,308**

Tabela 7 - Latência no processamento dos despachos

O tempo total despendido no carregamento dos despachos no *Druid*, é dado pela multiplicação entre a quantidade total de despachos e a latência média com que cada despacho é inserido. Por meio dessa multiplicação, foi possível determinar um valor aproximado para a latência no carregamento da totalidade dos despachos no *Druid*. Juntando aos outros valores de latência determinados para os restantes componentes desta arquitetura, obtém-se um valor um pouco acima de um minuto para o processamento de todos os despachos. Assim, pode-se aferir que o valor obtido para a latência no processamento dos despachos é inflacionado sobretudo pela extração, transformação e carregamento dos dados através do *Apache Nifi*, que representa mais de metade do valor total da latência.

Finalmente, importa referir que o cenário de alteração quase simultânea de todos os registos, como utilizado neste teste, é manifestamente irrealista de acontecer num cenário real de utilização. Assim, é expectável que os tempos de latência num ambiente de produção sejam substancialmente menores do que os aqui apresentados.

7. Conclusões

Tendo a arquitetura proposta toda implementada e após análise dos resultados, pode-se então fazer uma descrição dos problemas encontrados, das conclusões que podem ser retiradas da aplicação desta arquitetura, e dos trabalhos futuros que podem ser feitos numa perspetiva de continuidade e melhoramento da arquitetura.

7.1. Considerações gerais

O trabalho proposto teve como principal objetivo, diminuir o tempo de resposta das consultas de dados efetuadas através do SIJ, que requerem processamento de grandes volumes de dados. Para cumprir este objetivo foi desenhado e implementado uma arquitetura escalável, altamente disponível, com um consumo de dados em tempo real, idealmente sem custos e separado do ambiente de produção. Esta arquitetura é composta por um conjunto de soluções (ferramentas ou métodos), que executam funções que vão desde a deteção de uma alteração ou inserção de novos dados na base de dados do SIJ, até a respetiva publicação no *Data Warehouse* e posterior disponibilização desses dados para consultas.

Até a data atual no ambiente de produção, todas as consultas de dados feitas através do SIJ são executadas diretamente na base de dados operacional. Entretanto quando se trata de consultas que requerem processamento de grandes volumes de dados, o tempo de resposta normalmente aumenta, isto porque as bases de dados OLTP não foram projetadas para lidar com essas quantidades de informações. Por isso a solução proposta neste trabalho, vem no sentido de que grande parte dessas consultas, em vez de serem executadas na base de dados operacional, sejam executadas no *Data Warehouse* implementado. Já que o *Druid*

possui mecanismos de processamento de dados, como indexação e agregações que contribuem para que o tempo de resposta das *queries* seja muito baixo, o que pode ser comprovado pelos resultados de testes de *benchmarking* efetuados no capítulo 6. Além disso, todo o processo anterior a publicação dos dados no *Druid*, foi projetado para que assim que fosse detetado qualquer alteração dos dados armazenados na base de dados operacional, essa alteração fosse replicada para o *Druid*, passando por várias transformações.

Aproveitando a infraestrutura implementada para o *Data Warehouse*, foi também configurada uma ferramenta de visualização OLAP com o intuito de permitir análises com dados em tempo real. Esta ferramenta de visualização destaca-se pela facilidade de utilização, sendo bastante intuitiva o que facilita na construção de vários tipos de visualizações com os dados do SIJ provenientes do *Druid*. A principal finalidade para a utilização deste tipo de ferramenta de visualização, é permitir que as decisões possam ser tomadas de forma informada, ou seja com base nos dados apresentados.

Assim, foi possível provar que a arquitetura implementada é bastante eficiente no processamento de grandes quantidades de dados e consequentemente na disponibilização de informação. Portanto, quando esta infraestrutura for implementada no ambiente de produção prevê-se que tenha um impacto grande no desempenho das tarefas dos utilizadores, permitindo responder as consultas que requerem maior processamento de dados, de forma mais rápida em relação ao panorama atual do SIJ.

7.2. Problemas encontrados

Até os dias de hoje, não existe uma metodologia claramente definida para o desenvolvimento de *Data Warehouse*, principalmente quando o processamento de dados é feito em tempo real. A forma como os *Data Warehouse* tradicionais eram implementados deixou de ser valida, e obviamente isto traz desafios para a equipa responsável pela implementação deste tipo de infraestrutura. Para começar, as opções encontradas de sistemas de armazenamento *open source*, que se enquadram nos requisitos de um *Data Warehouse* em tempo real são muito reduzidas, e depois mesmo para as soluções existentes não se consegue encontrar documentação suficiente para casos de implementação pratica num ambiente de produção. Isto acontece porque apesar de ser uma área em grande

expansão, ainda é tudo bastante recente no que se refere a soluções para processamento de dados em tempo real.

Dos sistemas de armazenamento abordados ao longo do estado de arte que se encaixam no paradigma de *Data Warehouse* em tempo real, o *Druid* foi a solução adotada. A documentação sobre o *Druid* não é suficientemente transparente sobre como configurá-lo para um caso real de implementação, visto que isto depende muito dos requisitos e consequentemente dos recursos disponíveis nas máquinas onde a implementação esta a ser feita. Outra dificuldade encontrada está relacionada com a forma como as *queries* são efetuadas no *Druid*, utilizando uma estrutura proprietária no formato *json*, totalmente fora do padrão SQL normalmente utilizado. Em princípio, se fosse utilizada alguma das soluções de *Data Warehouse* comercial e em tempo real, seria possível evitar estes problemas apontados. No entanto um dos objetivos desta implementação é reduzir os custos o máximo possível.

7.3. Trabalhos futuros

Como referido anteriormente, os testes foram efetuados individualmente a cada componente da arquitetura implementada. Apesar de terem sido utilizados dados reais introduzidos por intermédio do SIJ, contudo os testes efetuados foram feitos num ambiente controlado, por um período limitado, e sem a interação dos utilizadores. Portanto, propõem-se que sejam efetuados testes com vários utilizadores em máquinas diferentes, por forma a recolher dados sobre o comportamento da arquitetura implementada em cenários reais de utilização. Depois, seria interessante implementar e configurar a arquitetura desenvolvida no ambiente de produção do SIJ, interagindo com os vários intervenientes da justiça que estão trabalhando diariamente nos seus processos.

8. Bibliografia

[1] Science Buddies, “The Engineering Design Process.” [Online]. Available: https://www.sciencebuddies.org/science-fair-projects/engineering-design-process/engineering-design-process-steps#keyinfo. [Accessed: 23-Jun-2018].

[2] R. Morais, J. Sousa Pinto, and C. Teixeira, “Sistema de Informação da Justiça de Cabo Verde (PDF Download Available),” *Revista do Ministério Público*, 2014. [Online]. Available: https://www.researchgate.net/publication/271510799_Sistema_de_Informacao_da_Justica_de_Cabo_Verde. [Accessed: 13-Dec-2017].

[3] C. Teixeira and J. Sousa, “Assisted On-Job Training,” *E-Learning - Eng. On-Job Train. Interact. Teach.*, no. March, 2012.

[4] J. Rosa, C. Teixeira, and J. Sousa Pinto, “Risk factors in e-justice information systems,” *Gov. Inf. Q.*, vol. 30, no. 3, pp. 241–256, Jul. 2013.

[5] R. Morais, P. J. Sousa, C. Teixeira, and S. Santos, “Sistema de Informação da Justiça de CaboVerde,” *Rev. do Ministério Público*, vol. 137, pp. 261–273, 2014.

[6] J. Vigário, “Utilização otimizada de recursos em testes de software na cloud,”

Aveiro, 2015.

[7] José Adelino Almeida Brazeta, “Testes de Software no Sistema de Informação da Justiça Cabo-Verdiana,” Aveiro, 2014.

[8] Microsoft, “Documentação do SQL Server | Microsoft Docs.” [Online]. Available: https://docs.microsoft.com/pt-br/sql/sql-server/sql-server-technical-documentation?view=sql-server-2017. [Accessed: 26-Jul-2018].

[9] Microsoft, “Entity Framework.” [Online]. Available: https://docs.microsoft.com/en-us/ef/. [Accessed: 26-Jul-2018].

[10] J. Gantz and D. Reinsel, “IDC IVIEW THE DIGITAL UNIVERSE IN 2020: Big Data, Bigger Digital Shadows, and Biggest Growth in the Far East,” 2012.

[11] “Business Intelligence - BI - Gartner IT Glossary.” [Online]. Available: https://www.gartner.com/it-glossary/business-intelligence-bi. [Accessed: 18-Dec-2017].

[12] Attunity, “CDC Change Data Capture | Attunity.” [Online]. Available: https://www.attunity.com/cdc-change-data-capture/. [Accessed: 23-Jul-2018].

[13] Microsoft, “About Change Data Capture (SQL Server),” 2017. [Online]. Available: https://docs.microsoft.com/en-us/sql/relational-databases/track-changes/about-change-data-capture-sql-server?view=sql-server-2017. [Accessed: 10-Jul-2018].

[14] Attunity, “Data Replication Tool | Attunity.” [Online]. Available: https://www.attunity.com/data-replication-tool/. [Accessed: 23-Jul-2018].

[15] Attunity, “The Architectural Principles of Attunity Replicate: Scalability & Flexibility.” [Online]. Available: https://www.attunity.com/blog/the-architectural-principles-of-attunity-replicate-scalability-flexibility/. [Accessed: 23-Jul-2018].

[16] E. Malinowski and E. Zimányi, “A conceptual model for temporal data warehouses and its transformation to the ER and the object-relational models,” *Data Knowl. Eng.*, vol. 64, no. 1, pp. 101–133, Jan. 2008.

[17] R. M. Bruckner, B. List, and J. Schiefer, “Striving towards Near Real-Time Data Integration for Data Warehouses,” Springer, Berlin, Heidelberg, 2002, pp. 317–326.

[18] D. Jin, T. Tsuji, and K. Higuchi, “An Incremental Maintenance Scheme of Data Cubes and Its Evaluation,” *Inf. Media Technol.*, vol. 4, no. 2, pp. 364–376, 2009.

[19] C. Gambino, D. Rice, J. Niemiec, M. Johnson, and J. Martz, *Apache NiFi for dummies*, Hortonwork. New Jersey: John Wiley & Sons, Inc., 2018.

[20] “Apache NiFi In Depth.” [Online]. Available: https://nifi.apache.org/docs/nifi-docs/html/nifi-in-depth.html. [Accessed: 06-Oct-2018].

[21] StreamSets, “Data Collector User Guide - Getting Started.” [Online]. Available: https://streamsets.com/documentation/datacollector/latest/help/. [Accessed: 11-Jul-2018].

[22] R. T. Fielding, “Architectural Styles and the Design of Network-based Software Architectures,” California, 2000.

[23] XMPP, “The most secure messaging standard.” [Online]. Available:

https://xmpp.org/. [Accessed: 17-Aug-2018].

[24] MQTT, “Documentation.” [Online]. Available: http://mqtt.org/documentation. [Accessed: 17-Aug-2018].

[25] STOMP, “The Simple Text Oriented Messaging Protocol.” [Online]. Available: https://stomp.github.io/. [Accessed: 17-Aug-2018].

[26] AMQP, “Advanced Message Queuing Protocol.” [Online]. Available: https://www.amqp.org/. [Accessed: 17-Aug-2018].

[27] J. Korab, *Understanding Message Brokers*, First Edit. Gravenstein Highway North: O’Reilly Media, Inc., 2017.

[28] Apache Kafka, “A distributed streaming platform.” [Online]. Available: https://kafka.apache.org/. [Accessed: 17-Aug-2018].

[29] J. Kreps, L. Corp, N. Narkhede, and J. Rao, “Kafka: a Distributed Messaging System for Log Processing,” 2011.

[30] “Apache Kafka.” [Online]. Available: https://kafka.apache.org/documentation/. [Accessed: 19-Jul-2018].

[31] Apache Kafka, “Documentation.” [Online]. Available: https://kafka.apache.org/10/documentation/streams/architecture. [Accessed: 11-Jul-2018].

[32] “Apache ActiveMQ ™ -- Index.” [Online]. Available: http://activemq.apache.org/. [Accessed: 18-Jul-2018].

[33] B. Snyder, D. Bosnanac, and R. Davies, *ActiveMQ in action*. Manning, 2011.

[34] “Apache ActiveMQ ™ -- Topologies.” [Online]. Available: http://activemq.apache.org/topologies.html. [Accessed: 18-Jul-2018].

[35] “RabbitMQ - Messaging that just works.” [Online]. Available: https://www.rabbitmq.com/. [Accessed: 19-Jul-2018].

[36] “RabbitMQ - What can RabbitMQ do for you?” [Online]. Available: https://www.rabbitmq.com/features.html. [Accessed: 19-Jul-2018].

[37] A. Videla and J. J. W. Williams, *RabbitMQ in action : distributed messaging for everyone*. Manning, 2012.

[38] J. Gray *et al.*, “Data Cube: A Relational Aggregation Operator Generalizing Group-By, Cross-Tab, and Sub-Totals,” *Data Min. Knowl. Discov.*, vol. 1, no. 1, pp. 29–53, 1997.

[39] M. Grund, J. Krüger, H. Plattner, A. Zeier, P. Cudre-Mauroux, and S. Madden, “HYRISE,” *Proc. VLDB Endow.*, vol. 4, no. 2, pp. 105–116, Nov. 2010.

[40] A. Kemper and T. Neumann, “HyPer: A hybrid OLTP&OLAP main memory database system based on virtual memory snapshots,” in *2011 IEEE 27th International Conference on Data Engineering*, 2011, pp. 195–206.

[41] Druid, “Interactive Analytics at Scale.” [Online]. Available: http://druid.io/. [Accessed: 26-Jul-2018].

[42] F. Yang, E. Tschetter, X. Léauté, N. Ray, G. Merlino, and D. Ganguli, “Druid,” in *Proceedings of the 2014 ACM SIGMOD international conference on Management of data - SIGMOD ’14*, 2014, pp. 157–168.

[43] The Apache Software Foundation, “Apache ZooKeeper - Home.” [Online]. Available: https://zookeeper.apache.org/. [Accessed: 18-Dec-2017].

[44] F. Li, M. T. Ozsu, G. Chen, and B. C. Ooi, “R-Store: A scalable distributed system for supporting real-time analytics,” in *2014 IEEE 30th International Conference on Data Engineering*, 2014, pp. 40–51.

[45] The Apache Software Foundation, “Apache HBase – Apache HBase™ Home.” [Online]. Available: https://hbase.apache.org/. [Accessed: 18-Dec-2017].

[46] J. Dean and S. Ghemawat, “MapReduce: Simplified Data Processing on Large Clusters,” *Commun. ACM*, vol. 51, no. 1, p. 107, Jan. 2008.

[47] F. Li, B. C. Ooi, M. T. Özsu, and S. Wu, “Distributed data management using MapReduce,” *ACM Comput. Surv.*, vol. 46, no. 3, pp. 1–42, Jan. 2014.

[48] D. Jiang, B. C. Ooi, L. Shi, and S. Wu, “The performance of MapReduce,” *Proc. VLDB Endow.*, vol. 3, no. 1–2, pp. 472–483, Sep. 2010.

[49] The Apache Software Foundation, “Welcome to Apache™ Hadoop®!” [Online]. Available: http://hadoop.apache.org/. [Accessed: 12-Dec-2017].

[50] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, “The Hadoop Distributed File System,” in *2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)*, 2010, pp. 1–10.

[51] “Rhombus.” [Online]. Available: https://github.com/Pardot/Rhombus. [Accessed: 26-Jul-2018].

[52] InfluxDB, “InfluxData (InfluxDB) | Time Series Database Monitoring & Analytics.” [Online]. Available: https://www.influxdata.com/. [Accessed: 13-Dec-2017].

[53] OpenTSDB, “OpenTSDB - A Distributed, Scalable Monitoring System.” [Online]. Available: http://opentsdb.net/. [Accessed: 13-Dec-2017].

[54] MonetDB, “The column-store pioneer | MonetDB.” [Online]. Available: https://www.monetdb.org/Home. [Accessed: 13-Dec-2017].

[55] Blueflood, “Blueflood DB.” [Online]. Available: http://blueflood.io/. [Accessed: 26-Jul-2018].

[56] “Newts.” [Online]. Available: http://opennms.github.io/newts/. [Accessed: 26-Jul-2018].

[57] “KairosDB.” [Online]. Available: https://kairosdb.github.io/. [Accessed: 26-Jul-2018].

[58] A. Bader, “Comparison of Time Series Databases,” University of Stuttgart, Stuttgart, 2016.

[59] A. Bader, O. Kopp, and M. Falkenthal, “Survey and Comparison of Open Source Time Series Databases.”

[60] Tableau, “O que é visualização de dados? Uma definição, exemplos e recursos.” [Online]. Available: https://www.tableau.com/learn/articles/data-visualization. [Accessed: 02-Jul-2018].

[61] Apache Superset, “Apache Superset (incubação) - Documentação do Apache Superset.” [Online]. Available: https://superset.incubator.apache.org/. [Accessed: 02-Jul-2018].

[62] Grafana, “The open platform for analytics and monitoring.” [Online]. Available: https://grafana.com/. [Accessed: 18-Aug-2018].

[63] Metabase, “Metabase is the easy, open source way for everyone in your company to ask questions and learn from data.” [Online]. Available: https://www.metabase.com/. [Accessed: 18-Aug-2018].

[64] V. Ogievetsky, “Pivot: A Fast Data Exploration UI for Druid,” *Imply*, 2015. [Online]. Available: https://imply.io/post/hello-pivot. [Accessed: 03-Jul-2018].

[65] Imply, “Operational analytics for your business.” [Online]. Available: https://imply.io/. [Accessed: 18-Aug-2018].

[66] Airbnb, “Airbnb Engineering & Data Science.” [Online]. Available: https://airbnb.io/. [Accessed: 18-Aug-2018].

[67] A. Scott, B. Kyryliuk, E. Brumbaugh, J. Feng, M. Beauchemin, and V. Liu,

"Superset: Scaling Data Access and Visual Insights at Airbnb," *Airbnb Engineering & Data Science*, 2017. [Online]. Available: https://medium.com/airbnb-engineering/superset-scaling-data-access-and-visual-insights-at-airbnb-3ce3e9b88a7f. [Accessed: 02-Jul-2018].

[68] Apache, "Welcome to The Apache Software Foundation!" [Online]. Available: https://www.apache.org/. [Accessed: 18-Aug-2018].

[69] Armin Ronacher, "Flask (A Python Microframework)." [Online]. Available: http://flask.pocoo.org/. [Accessed: 18-Aug-2018].

[70] Grafana Labs, "Grafana documentation | Grafana Documentation." [Online]. Available: http://docs.grafana.org/. [Accessed: 03-Jul-2018].

[71] Metabase, "Metabase Documentation." [Online]. Available: https://www.metabase.com/docs/v0.29.3/. [Accessed: 04-Jul-2018].

[72] S. Al-Sakran, "We put questions inside your questions," *Metabase*, 2017. [Online]. Available: https://www.metabase.com/blog/Nested-Queries/. [Accessed: 04-Jul-2018].

[73] Druid, "Indexing Service." [Online]. Available: http://druid.io/docs/latest/design/indexing-service.html. [Accessed: 09-Aug-2018].

[74] Druid, "Kafka Indexing Service." [Online]. Available: http://druid.io/docs/latest/development/extensions-core/kafka-ingestion.html. [Accessed: 09-Aug-2018].

[75] B. F. Cooper, “Yahoo Cloud Serving Benchmark | research.yahoo.com,” 2010. [Online]. Available: https://research.yahoo.com/news/yahoo-cloud-serving-benchmark/. [Accessed: 13-Oct-2018].

9. Anexos

Neste capítulo são apresentadas informações de suporte a implementação da arquitetura desenvolvida, comportando informações sobre a configuração do *Deep Storage* com HDFS, e alguns scripts de automação utilizados.

9.1. Configuração do Deep Storage com HDFS

Neste trabalho o *Deep Storage* foi configurado para armazenamento local em detrimento das outras opções suportadas pelo *Druid*, por se entender que nesta fase, este tipo de armazenamento atende aos requisitos do sistema implementado. Contudo, durante o período dedicado a escolha das soluções que melhor se adequam aos requisitos estabelecidos para implementação desse sistema, houve uma altura em que foi feito uma experiencia simples de utilização do *Druid* com o *Deep Storage* configurado com o *Hadoop Distributed File System* (HDFS). Deste modo, ter o *Hadoop* instalado e configurado com um *cluster* de um único nó é suficiente para montar toda a infraestrutura necessária para este sistema de ficheiros. Depois é necessário que o *Druid* seja corretamente configurado para utilizar o HDFS como o sistema de armazenamento permanente (*Deep Storage*). Por padrão, a extensão *druid-hdfs-storage* relativo ao HDFS já se encontra no diretório das extensões do *Druid*, bastando então adiciona-lo a propriedade *druid.extensions.loadList* do *common.runtime.properties* e que os ficheiros de configuração XML (*core-site.xml*, *hdfs-site.xml*, *yarn-site.xml*, *mapred-site.xml*) sejam copiados para o *classpath* do *Druid*. Isso permite que o *Druid* encontre o *cluster Hadoop* e submeta as tarefas de forma correta.

9.2. *Scripts de Automação*

Neste tópico, são ilustrados os scripts de automação para iniciar os nós *coordinator*, *historical*, *overlord*, *middle manager* e *broker*. Também é apresentado o script utilizado para executar as especificações de configuração dos serviços de indexação para cada fonte de dados.

9.2.1. Execução do Druid

Antes de iniciar o consumo de dados no *Druid*, primeiramente é preciso garantir que todos os nós (*coordinator*, *overlord*, *historical*, *middlemanager* e o *broker*) configurados para este sistema estejam funcionando. Na Figura 39 é apresentado o script utilizado para colocar o *Druid* em funcionamento por intermedio da execução dos seus nós.

```
sudo nohup java `cat conf/druid/coordinator/jvm.config | xargs` \
-cp conf/druid/_common:conf/druid/coordinator:lib/* io.druid.cli.Main server coordinator \
2>&1 > /home/oneadmin/druid-log/coordinator.log &

sudo nohup java `cat conf/druid/overlord/jvm.config | xargs` \
-cp conf/druid/_common:conf/druid/overlord:lib/* io.druid.cli.Main server overlord \
2>&1 > /home/oneadmin/druid-log/overlord.log &

sudo nohup java `cat conf/druid/historical/jvm.config | xargs` \
-cp conf/druid/_common:conf/druid/historical:lib/* io.druid.cli.Main server historical \
2>&1 > /home/oneadmin/druid-log/historical.log &

sudo nohup java `cat conf/druid/middleManager/jvm.config | xargs` \
-cp conf/druid/_common:conf/druid/middleManager:lib/* io.druid.cli.Main server middleManager \
2>&1 > /home/oneadmin/druid-log/middleManager.log &
```

Figura 39 - Script utilizado para colocar o Druid em funcionamento

9.2.2. Serviço de Indexação Kafka

Depois que o *Druid* estiver em execução, já é possível iniciar os serviços de indexação. Em concreto, para este trabalho foram submetidas especificações de serviços de indexação para três fontes de dados, os quais por sua vez darão origem a três supervisores, um para cada fonte de dados (Figura 40).

```
curl -X POST -H 'Content-Type: application/json' \
-d @conf/tranquility/auto/auto-index-kafka.json http://localhost:8090/druid/indexer/v1/supervisor

curl -X POST -H 'Content-Type: application/json' \
-d @conf/tranquility/despacho/despacho-index-kafka.json http://localhost:8090/druid/indexer/v1/supervisor

curl -X POST -H 'Content-Type: application/json' \
-d @conf/tranquility/requerimento/requerimento-index-kafka.json http://localhost:8090/druid/indexer/v1/supervisor
```

Figura 40 - Shell scritp para iniciar os serviços de indexação Kafka